

IBM Data Virtualization Manager for z/OS

Doug Dailey

Guillaume Arnould

Marie-Therese Bouedo

Willem de Gelder

Francesco Borrello

Dave Trotter

Nasser Ebrahim

Jonathan Sloan

John Casey

Bill Powers

Shawn Sullivan

Robin Ramadil-Kannan

Rajesh Sambandhan

Mahesh Sugavanam

Coreen Wilson

Jeff Lutzow



 Analytics



IBM Redbooks

IBM Data Virtualization Manager for z/OS

October 2021

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (October 2021)

This edition applies to IBM Data Virtualization Manager for z/OS Version 1.1.0.

© Copyright International Business Machines Corporation 2021. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
Authors	xii
Now you can become a published author, too!	xiv
Comments welcome	xiv
Stay connected to IBM Redbooks	xiv
Chapter 1. Introduction	1
1.1 Protecting your investment by using IBM Z technology	2
1.2 Why DVM for z/OS in modernization?	2
1.3 Why is today different?	4
1.4 What does the market offer today?	5
1.5 What can you do?	5
1.6 Resolving the data latency gap through data virtualization	6
1.7 DVM for z/OS	6
1.8 IBM Cloud Pak for Data	6
1.9 Unlock enterprise data for virtually any application	7
1.10 Why and when should you consider DVM for z/OS	8
Chapter 2. Architecture and implementation	9
2.1 Reference architecture for DVM for z/OS	10
2.2 Technical components	10
2.3 SQL engine and query optimization	12
2.3.1 Query processing by SQL engine	12
2.3.2 WHERE predicate PUSHDOWN	12
2.3.3 Join processing	12
2.3.4 Referential integrity	13
2.3.5 Virtual Parallel Data	13
2.3.6 Flatten arrays	14
2.4 Metadata repository	14
2.4.1 DVM catalog tables	15
2.5 Parallel processing through MapReduce	16
2.6 z/OS resident optimization	17
2.7 zIIP eligibility and data compression	18
2.8 DVM Studio	19
2.9 Integrated DRDA Facility	21
2.9.1 Peer-to-peer configuration	21
2.9.2 Db2 Information Hub	22
2.10 DVM endpoint connections	23
2.10.1 Drivers	23
2.10.2 DVM Parser and Data Mapping Facility	24
2.10.3 DS-Client API interface	25
2.10.4 z/OS Connect Enterprise Edition	26
2.10.5 Java Database Connectivity Gateway	26
2.10.6 Connection and port security	27
2.11 Summary	28

Chapter 3. Installation and configuration	29
3.1 Installation overview	30
3.2 Creating the DVM server data sets	31
3.3 Setting up the security application	33
3.4 Configuring the Workload Manager	34
3.5 Authorizing the program LOAD library	34
3.6 Creating a backup of the product libraries	35
3.7 Configuring support for the DBCS system	36
3.8 Customizing the DVM server for access to databases	36
3.8.1 Customizing the relational database	37
3.8.2 IMS database customization	38
3.8.3 Adabas customization	39
3.8.4 Configuring the started task JCL	39
3.8.5 Configuring the Command List	41
3.9 Verifying the installation	43
Chapter 4. Connecting to z/OS data sources	45
4.1 Introduction	46
4.2 Getting started.	47
4.3 Direct access to z/OS databases	47
4.3.1 ADABAS	47
4.4 Db2 for z/OS	52
4.4.1 Db2 for z/OS access options	53
4.5 IBM ESA/IMS database	54
4.5.1 IMS database control	54
4.5.2 IMS Direct	54
4.5.3 IMS Open Database Access	55
4.5.4 Configuring IMS	55
4.5.5 Creating virtual tables	55
4.5.6 Enabling IMS Direct	55
4.6 Accessing mainframe files	56
4.6.1 VSAM	57
4.6.2 System and operations logging	60
4.6.3 Delimited file data sets	64
4.6.4 System Management Facility	66
4.6.5 Db2 unload data sets	74
Chapter 5. Connecting to non-Z data sources	79
5.1 Introduction	80
5.1.1 Standard access to data sources	80
5.1.2 Distributed Relational Database Architecture	80
5.2 Accessing non-z/OS data sources by using the JDBC Gateway server	81
5.2.1 Setting up the JDBC Gateway server	82
5.2.2 Installing the JDBC Gateway server	83
5.2.3 Running JDBC Gateway server by using UNIX System Services	83
5.2.4 Managing JDBC Gateway server software upgrades	85
5.2.5 Starting the JDBC Gateway server that uses administrative UI	86
5.2.6 Configuring data sources that use the JDBC Gateway server UI	87
5.2.7 Configuring the DVM server to access the JDBC Gateway server	88
5.2.8 Setting user credentials for the JDBC Gateway server	90
5.2.9 Establishing secure access using AVZDRATH	90
5.2.10 User access that uses rules	91
5.2.11 Using rules to ensure global user authorization	92
5.2.12 Connecting to a JGATE Data Source in DVM Studio	94

5.2.13 Secure access that uses AVZDRATH.	100
Chapter 6. Access methods	103
6.1 Interface methods for client access.	104
6.2 Standard access	105
6.2.1 JDBC/ODBC (including security or Kerberos).	105
6.2.2 ODBC (including security or Kerberos).	106
6.2.3 Java application programming interface	107
6.3 DS Client.	109
6.3.1 CICS and other TXN or workload balancers.	109
6.3.2 Using Data Virtualization Manager in a COBOL program.	110
6.4 REST and SOAP Web service interfaces	111
6.4.1 IBM z/OS Connect Enterprise Edition.	111
6.4.2 Configuring the DVM server for use with z/OS Connect.	111
6.4.3 Installing the DVM Service Provider	113
6.4.4 Creating zCEE RESTful APIs for access to the DVM server	115
6.4.5 Db2 Query Management Facility API	120
6.5 Integrated Data Facility for mainframe applications	122
6.5.1 DVM server subsystem in the Db2 communications database.	122
6.5.2 Use cases	122
6.5.3 Choosing Db2 UDTF or IDF	123
6.6 Db2 for z/OS UDTF.	123
6.7 Db2 federation.	126
6.8 IBM Cloud Pak for Data	128
6.8.1 Cloud Pak for Data interface and adding a DVM connection	128
6.8.2 Previewing data from your newly connected DVM server.	130
Chapter 7. Managing and monitoring	131
7.1 Accessing the ISPF interface	132
7.1.1 DVM server ISPF panel	132
7.1.2 Creating virtual tables in the ISPF interface	135
7.1.3 ISPF interface and IBM Parallel Sysplex	139
7.2 DVM Studio.	139
7.2.1 Navigator wizard	140
7.2.2 DVM Studio perspectives and views.	144
7.2.3 Common tools.	147
7.2.4 More menu options	150
7.2.5 Using DVM Studio to virtualize IMS data segments	151
7.3 Batch interface	161
7.3.1 Creating a virtual table with the batch interface	161
7.3.2 Migrate virtual tables with the batch interface.	162
7.3.3 Querying virtual tables with the batch interface	164
7.4 API interface	164
7.4.1 API interface purpose	164
7.4.2 Calling the API interface	165
7.4.3 API functions.	165
7.4.4 API interface and DVM Studio	166
7.5 Metadata	166
Chapter 8. Performance tuning and query optimization	167
8.1 Introduction	168
8.2 Combined GP and zIIP consumption	168
8.3 Parallel I/O and MapReduce.	170
8.4 Virtual Parallel Data	171

8.4.1	Using VPD groups	171
8.4.2	Example	171
8.4.3	Considerations and limits	172
8.5	Workload management	172
8.5.1	Configuring WLM for the DVM server	173
8.5.2	Working with multiple DVM servers	174
8.5.3	Load balancing with CICS regions	177
8.5.4	Db2-Direct and IMS-Direct	177
8.5.5	Java Database Connectivity performance	180
8.6	ODBC performance	185
8.7	Integrated Data Facility and DS Client API	185
8.8	Query optimization and performance	186
8.8.1	SQL best practices	186
8.8.2	Performance testing with Apache JMeter	191
8.8.3	Performance testing by using the Command Line Tester	192
Chapter 9.	Capacity planning and deployment	195
9.1	Capacity planning	196
9.2	Monitoring workloads	199
9.2.1	Monitoring capacity with SMF records	199
9.2.2	Monitoring performance by using SMF Record 249 Subtype 06	200
9.2.3	Monitoring by using DVM ISPF panels	201
9.3	Capacity planning for future growth	202
9.3.1	Customer example	202
9.3.2	General recommendations	202
9.4	Scaling for growth with the DVM server	203
9.4.1	Memory consumption	203
9.4.2	zIIP processing	204
9.4.3	Use of MapReduce for parallelism	204
9.4.4	Performance differences by access method	205
9.5	Workload balancing	206
9.6	User concurrency	206
9.7	Best practices for deploying the DVM server	207
9.7.1	Data sources	207
9.7.2	Naming conventions	207
9.8	Developing queries	208
9.8.1	Creating virtual tables	208
9.8.2	Combining data from different data sources	208
9.8.3	Creating a query	208
9.8.4	Testing	209
9.8.5	Embedding your query in an application	209
9.9	Administering the DVM server in production	211
9.9.1	Limiting access to the DVM server	211
9.9.2	High availability configurations	211
Chapter 10.	Best practices for project success	213
10.1	Defining successful projects	214
10.2	Defining the approach	214
10.3	POC checklist	216
10.3.1	Timelines	216
10.3.2	Setting scope	216
10.3.3	Focus	216
10.3.4	Success criteria	217

10.3.5 Best practices	217
10.3.6 Roles and responsibilities	218
10.3.7 Installation	219
10.3.8 Configuration	219
10.3.9 Architectural topology	219
10.3.10 Defining use cases	220
10.3.11 Best practices for defining success criteria	221
10.3.12 Concluding the POC	221
10.3.13 Finalizing deliverables	221
Appendix A. Project survey	225
Business drivers	226
Virtualization topology	227
Distance between the DVM servers and data sources	227
Primary use cases	227
Physical storage or memory	228
Environment	229
Hardware configuration	229
z/OS environment	229
Access to data sources	230
Client connections to DVM for z/OS	230
Data sources	230
Application workloads	231
Appendix B. Java API sample code snippet	233
Available metadata in the DVM server	234
Appendix C. Troubleshooting and diagnosing	239
Initially characterizing problems	240
Must Gather information	240
DVM for z/OS version	240
High Module date	240
PTF Level	240
Operating system	241
DVM server environment	241
Capturing trace browse from the DVM server ISPF panel	241
Capturing a copy of the Trace Browse	241
Displaying and viewing server traces	243
Capturing and printing server trace output	244
Using DVM Studio to display and view a server trace	245
Using DVM Studio to diagnose SQL results	245
Consolidating server trace over multiple DVM servers	246
Search Techdocs for answers	248

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

CICS®	IBM z Systems®	Redbooks (logo)  ®
Cognos®	IBM z13®	SPSS®
Db2®	IMS/ESA®	WebSphere®
DB2®	Informix®	z Systems®
IBM®	Netezza®	z/OS®
IBM Cloud®	Parallel Sysplex®	z13®
IBM Cloud Pak®	RACF®	z15™
IBM Watson®	Rational®	zEnterprise®
IBM Z®	Redbooks®	

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication presents an overview of the IBM Data Virtualization Manager for IBM z/OS® offering and the role it plays in accessing traditional non-relational data on IBM Z.

If there is anything true about the IT industry, it is change and data. Data Virtualization Manager for z/OS is built with both of these absolute truths in mind. With Data Virtualization Manager for z/OS, an organization can extend its infrastructure and data investments to support new ways of accessing and presenting data within modern applications. Modernizing access to those highly valuable traditional non-relational data sources is its reason for existence.

The book begins with general concepts, the value of virtualization to an organization, and the benefits of virtualizing data assets that originate on IBM Z®. It compares the benefits and implications of accessing data where it originates to the effect of moving that data to another platform for data access. Modernization is often at the center of any Data Virtualization project and Data Virtualization Manager for z/OS can play a key role in these efforts.

Next, the book describes the architecture of the Data Virtualization Manager for z/OS server and provides technical details of the data asset virtualization process so that technical users can better prepare for deployment of this new technology. The book provides a description of the server's most important components and functions so that a user can better understand how to take advantage of the extensive capabilities.

Beyond the fundamental description of the server components, the book documents how to install and configure the server, how to connect it to various data sources (such as VSAM and IMS), the different access methods for different types of data sources, and how to manage, monitor, and tune. A chapter about capacity planning and deployment also is included, along with best practices for project success. Lastly, some helpful appendixes are available, including one about troubleshooting and diagnostics.

Although this book does not include all the documentation that is relevant to Data Virtualization Manager for z/OS, we believe it to be a good roadmap to get you on your way to a successful virtualization project.

The introductory chapters of this publication are intended for a broad technical audience, including IT system architects, technical IT managers, operations managers, system programmers, application developers, and other technical roles. The subsequent chapters provide more technical details about the virtualization of specific types of data sets on IBM Z and might be more suitable to technicians who are implementing virtualization on those specific data assets.

Authors

This book was produced by a team of specialists from around the world, including specialists from both IBM and Rocket Software, Inc.

Doug Dailey is an IBM Product Manager with over 10 years of experience in building software and solutions for customers. Doug started working in technical support for IBM Informix® Software (now an IBM company) and transitioned to a Technical Account Manager role and soon thereafter a CSM Manager for IBM's Accelerated Value Program. His passion is helping customers win. For over 10 years, Doug was a product manager at Informix Software, IBM Netezza® and Pure Data for Analytics, and IBM Db2 Replication for Continuous Availability. His specialization is data federation and Data Virtualization technologies and currently is the Product Manager for IBM Data Virtualization Manager for z/OS.

Guillaume Arnould joined IBM in 1996 and started in IBM z System Manufacturing Test Engineering before spending 2 years in Poughkeepsie, NY. In 2001, he joined the IBM Client Center in Montpellier to work as a Performance Expert on Db2 for z/OS client benchmarks. After 10 years as the Technical Team Lead in the Smarter Banking Showcase and engaging with customers. Guillaume is now an Advanced Technical Sales Expert and leads the Data & AI on IBM Z Solutions team. He is working on IBM Db2 Analytics Accelerator, Watson Machine Learning on Z, Db2 for z/OS, IBM Db2 Data Gate, and IBM Data Virtualization Manager for z/OS.

Marie-Therese Bouedo started in Information Technology programming PL2, COBOL, ASM, and other mainframe languages. She then worked as a system programmer for 18 years and as a database specialist for Db2z, Oracle, Information, and other database management systems. Marie then split the next 14 years working for IBM GTS as an infrastructure architect and in Technical Sales for IBM Z software with specialization in CICS®, z/OS Connect, and IBM Data Virtualization Manager for z/OS.

Willem de Gelder started as a COBOL/Db2® programmer in 1990. Since then, he has worked in several roles in development, operations, and consultancy. Always with Data & AI software on IBM® Mainframe. Currently, Willem works as a zHybrid Cloud pre-sales technical seller for IBM in Latin America.

Francesco Borrello Francesco Borrello is a z Data & AI Technical Sales Specialist in IBM Technology, Italy. He joined IBM in 2011 and obtained a master's degree in Centralized System for Cloud Computing. His main mission is to help customers in driving new business opportunities by adopting traditional Hybrid Data Management and Analytics and innovative ML/AI solutions on IBM Z®. His areas of specialty are in Db2 for z/OS, Db2 Tools, Db2 Analytics Accelerator, Data Replication, and Data Virtualization Manager for z/OS. He also was a presenter at several international conferences and technical user groups.

Dave Trotter began working as a developer and systems programmer in his early career for Db2 for IBM iSeries. Currently, he works as a technical specialist and technical seller for IBM System Z software solutions. Dave's area of specialty is guiding customers on analytics strategies and the use of the Db2 Analytics Accelerator for z/OS and Data Virtualization Manager for z/OS in their enterprise.

Nasser Ebrahim is a Solution Architect with IBM Systems Lab, focusing on data and AI solutions on IBM Z. In his role, Nasser helps Global System Integrators and clients to build solutions on data and AI technologies. He has 22 years of experience with IBM Z technologies, which includes application programming, system programming, Java run times, analytics, and machine learning. He was Java Current Release Technical Leader with IBM Software Lab before taking his current role as a solution architect.

Jonathan Sloan is a portfolio product marketing manager who focuses on IBM Data & AI products for the IBM Z mainframe platform. He excels in helping others understand how to apply advanced analytic and machine learning technology to business problems. Jonathan has experience in several industries with a focus on health care, insurance, financial services, and consumer packaged goods. He is passionate about helping organizations drive greater insight and value from enterprise data. He excels in working directly with customers and providing leadership within team environments.

John Casey is a Principal Solutions Advisor for Rocket Software. Inc., where he supports customers that use IBM's Db2 tools and IBM Data Virtualization Manager on z/OS on System Z. Before joining Rocket Software. Inc, John spent 20 years at IBM as a Field Technical Specialist on z/OS and Db2 related products.

Bill Powers has been working with IBM Information Management solutions since 1985. As an IBM customer for 36 years, Bill worked in application development, database administration, quality assurance, and system programming. His experience has spanned a range of industries, including health care, insurance, manufacturing, transportation, software, and public utilities. He has been working with Db2 for the last 29 years in the areas of database design and administration, performance and tuning, backup and recovery, installation, and migration. Bill works for Rocket Software, Inc., as a Senior Solutions Advisor.

Shawn Sullivan began working as a trainer and eventual product expert in Db2 QMF and Db2 Tools in 1998. He has been a QMF specialist for 23 years for cross-platform mainframe and distributed systems. Recently, he joined the Data Virtualization Manager for the z/OS team and the IBM Multi-factor Authentication team.

Robin Ramadil-Kannan started as an application programmer in z/OS and worked on Db2-relational, IMS-hierarchical, and IDMS-network databases for several Fortune 500 customers. He has since specialized in application re-writes, using Service Oriented Architecture and ETL space. Robin currently focuses on Data Virtualization technology for the mainframe.

Mahesh Sugavanam is a Senior Software Engineer for Rocket Software, Inc. Over the previous 22 years, Mahesh specialized in information management products for IBM system Z, Linux, UNIX, and Windows. He specialized in mainframe modernization and workload optimization and is extensively knowledgeable about Db2z over system, administration, data replication, and performance categories. Mahesh graduated 1st in class with a Master's degree in Computer Applications from Bharathidasan University in India.

Rajesh Sambandhan is a software engineer at Rocket Software, Inc., and has been in the Information Technology industry for more than 20 years. He primarily worked on the Java stack, databases, and ODBC/JDBC driver development. His area of focus has been Java technologies, test-driven development, application architecture and security, Web stack, and data science. He is currently leading the development of client components for IBM Data Virtualization Manager for z/OS.

Coreen Wilson spent nearly 10 years becoming a subject matter expert on the top cybersecurity frameworks during an era of vast security breaches, malware, and ransomware promoting the importance of cyber security to the C-suite. For the past two years, Coreen has been leading product marketing for IBM Z Systems at Rocket Software, Inc., championing open mainframe transformation through innovative technology, such as OSS/Zowe, ML/AI, and Data Virtualization.

Jeff Lutzow is a Senior Customer Service Engineer at Rocket Software, Inc. and has been in the Information Technology industry working on z/OS systems for more than 30 years. He has worked in application development, database administration, and product management in the Insurance and Health Care industries. He has worked in support of IBM Data Virtualization

Manager for z/OS since the product's inception. He is currently leading the services initiative for IBM Data Virtualization Manager for z/OS.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks® residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Introduction

This chapter introduces key concepts and value propositions for data virtualization for mainframe data sets. The chapter espouses the benefits of the use of IBM Data Virtualization Manager for z/OS (DVM for z/OS or just DVM) over various topologies, details key differentiators around cost savings and performance, along how this technology drives the what, how, and why conversation.

This chapter includes the following topics:

- ▶ 1.1, “Protecting your investment by using IBM Z technology” on page 2
- ▶ 1.2, “Why DVM for z/OS in modernization?” on page 2
- ▶ 1.3, “Why is today different?” on page 4
- ▶ 1.4, “What does the market offer today?” on page 5
- ▶ 1.5, “What can you do?” on page 5
- ▶ 1.6, “Resolving the data latency gap through data virtualization” on page 6
- ▶ 1.7, “DVM for z/OS” on page 6
- ▶ 1.8, “IBM Cloud Pak for Data” on page 6
- ▶ 1.9, “Unlock enterprise data for virtually any application” on page 7
- ▶ 1.10, “Why and when should you consider DVM for z/OS” on page 8

1.1 Protecting your investment by using IBM Z technology

With all the hype around new technology, cloud platforms, and merging development approaches, you might be wondering how IBM Z and the transactional data that is on it fit into this changing landscape.

IBM recognizes the importance of strengthening the partnership between IT and the business. The optimal position for today's IT organization is to be a full-fledged partner with lines of business to drive revenue and deliver on an organization's strategy. Therefore, IT organizations must change and transform along with their systems.

Many organizations say they want to modernize and transform, but the transformation is not only an end goal, it is a journey. To transform requires agility, which is a continuous process. You need technology that can continuously adapt to new business requirements and IT methods.

IBM Z has proven to be adaptable throughout its history and continues to introduce capabilities that adapt, respond, and lead, as new needs arise. New capabilities that are delivered on IBM Z integrate with IBM Cloud Pak for Data, IBM's strategic platform for Data and AI.

IBM Cloud Pak for Data is a fully integrated data and AI platform that modernizes how businesses collect, organize, and analyze data and infuse AI throughout their organizations. Integration between IBM Z and IBM Cloud Pak for Data readily takes advantage of your IBM Z data and resources.

Your IBM Z infrastructure investment holds significant value and can help your organization deliver continued business differentiation. Some might say that IBM Z and hybrid cloud is an unconventional combination. However, to paraphrase John Maynard Keynes to succeed, sometimes it is necessary to do so unconventionally rather than be like everyone else and perform conventionally. IBM Z understands this and offers the differentiated organization a differentiated platform.

This book discusses how IBM Z technology can complement and extend your existing and future hybrid cloud environment.

1.2 Why DVM for z/OS in modernization?

DVM for z/OS was released at the end of 2017. Since then, the market for data virtualization technology increased significantly. Several trends contribute to the recognition that data virtualization is a must-have component for many enterprise organizations.

The growth in corporate transactional data and sources, Hadoop, IoT data, the acceptance of cloud as an enterprise data store, and the availability and utility of public cloud services providing unique data are just some of the reasons that contribute to the continuing exponential expansion of data volume and type. With data being generated and stored everywhere, it is impossible to centralize it all and maintain its accuracy. Therefore, it is imperative that data must be accessed where it is stored or originates.

Within your enterprise infrastructures, you likely developed several to many large data repositories across several different vendor technologies. Each data repository features its own unique capabilities and value.

Each data repository vendor likely tells you the same thing: that is, that you should move (copy) all of your data to their platform. However, from a cost, efficiency, regulatory, security, and practical perspective, this advice does not make sense. Every copy of data has its own cost, latency, and risk. Copying all your data to wherever it might be needed is expensive, untimely, and presents potential regulatory, governance, and security risk issues.

Several years ago, the term *data gravity* was offered as an analogy to the concept of gravity in physics. Data gravity implies that data has mass, which attracts other objects as physical objects do. The more data there is, the greater the mass, the more likely it is to attract other objects, such as applications, services, and other data.

Although the cloud often uses a great deal of mind share, most essential corporate data is still generated behind the firewall. That data is typically transactional data, corporate treasure; therefore, data gravity applies to the typical organization's enterprise transactional servers. This is where the mainframe enters this expansive data story.

The mainframe is core to any organization's enterprise infrastructure. Although we think that the mainframe services only the largest organizations, many smaller organizations benefit from its security, resiliency, low per-transaction cost, virtualization, high performance, and unique value. For many of these organizations, the mainframe is their primary transactional platform.

According to IBM studies, 70 - 80% of all data originates behind the firewall and much of that originates on IBM Z. For many organizations, their transactional systems stayed largely unaltered for decades.

Upgrades focused on the latest versions of software to stay up to date in line with support requirements, fine-tuning of applications, and small changes to applications to keep up with the times. Government agencies, banks, insurance companies, and financial services organizations stayed with what served them well. Why change if something works? Today, organizations are under pressure to better serve their customers and bring new value to their constituents.

It is on the mainframe where unique data sources, transactional applications, data gravity, and modernization combine to offer an incredible opportunity for DVM for z/OS. DVM for z/OS offers organizations the potential to modernize the way they develop applications, simplify access to traditional non-relational IBM Z data sources, provide access to an extensive number of data sources (more than 35), and move from the 20th to the 21st century.

Wholesale application rewrites are generally not feasible, nor do they always bring significant differentiation from established technology. When an organization invests in differentiated business processes that are built on differentiated applications, it is to its advantage to continue to use those applications and the data that is stored in them. DVM for z/OS facilitates modernization efforts, which allows an organization to simplify access, reduce cost, and modernize interfaces.

This book focuses on the technology and capabilities that are offered by DVM for z/OS that make modernization easier. This book is somewhere between a getting started guide and a road map to get you where you want to go. You learn much about what DVM for z/OS can do for you and how you can do it.

Data from established applications became the lifeblood of new analytics, cloud, and mobile applications. Access to real-time transactional IBM Z data is quickly becoming a differentiating factor in support of these new applications:

- ▶ **Analytics**

According to market-leading analysts, insight-driven organizations are growing faster than their competitors, better-retaining customers and delivering significantly better returns on their investments. Insight-driven organizations use analytics to embed insight within business processes and software.

- ▶ **Cloud**

Cloud approaches to development, maybe more than the cloud service providers, are offering new techniques for application development, such as containerization and microservices. Organizations that use these new techniques are finding they offer better flexibility as compared to traditional monolithic, waterfall approaches to development. Cloud development techniques that lead to greater agility are an underlying foundation of disruption that the industry labeled digital transformation.

- ▶ **Mobile**

Although mobile has been around for some time, it is finally living up to its promise through the combination of analytics and cloud. Analytics are allowing organizations to drive greater value from their customer's mobile use. Cloud is allowing organizations limitless capacity to process data and use digital, momentary opportunities and provide those opportunities through their customer's mobile devices. Mobile devices largely became the de-facto interface with which customers interact with providers. A significant percentage of IBM Z workloads is driven by mobile applications, such as online banking and online purchases.

- ▶ **Modernization and digital transformation**

Analytics, cloud, and mobile are combining to completely rewrite industries and the rules of customer engagement. Digital transformation is offering unprecedented opportunities for organizations to improve customer relationships, acquire new customers, and grow wallet share. Organizations are innovating with new ways to take advantage of their enterprise data assets and engage with customers.

1.3 Why is today different?

Many organizations' IT architectures and the transaction systems, insight, and decision-making processes they support were designed decades ago. Traditional architectural approaches that depend solely on data movement can impede the ability to rapidly adapt to changing business cycles and adopt new development methods.

To deliver modern applications, organizations must:

- ▶ Facilitate and simplify access to relational and non-relational IBM Z transactional data
- ▶ Access and update live IBM Z data by way of modern APIs, such as SQL and RESTful (when combined with z/OS Connect)
- ▶ Reduce the cost and delay of moving data to non-IBM Z platforms

Data, its use, and the insights it provides feature a shelf-life. Some decisions require the most current data and real-time insight (at the point of interaction or transaction), which improves decision making in areas, such as fraud detection, up-sell and cross-sell efforts, and supporting real-time opportunities, such as digital moments.

New architectural patterns must consider a multi-speed approach to data delivery, the use of data, and a model that supports structured and semi-structured data across multiple disparate platforms. As your organization modernizes and takes advantage of new application development techniques, consider minimizing data movement as part of the modernization process. This minimization allows you to make the best use of your data as it exists in your operational systems.

Completely separating the systems of record from modern systems of engagement can result in customer defections, increased risk, and lost opportunity for improved operational productivity. We all have been frustrated by customer-facing applications that we know do not reflect an organization's current inventory or the status of a reservation.

Digital opportunities are momentary. They must be based on the exact real inventory status to make offers to customers. This approach is different than traditional approaches that use extensive data movement. Doing nothing (that is, merely maintaining these existing approaches) opens the door for the competition to disrupt your business and attract your customers.

New business with increased data volumes leads to higher resource usage and exacerbates an already challenging issue. Throwing hardware at the problem just pushes the problem off to another day. A change in approach is needed.

1.4 What does the market offer today?

The IT market is offering more alternatives than ever before for organizations to become more data-driven. It is important to take advantage of these innovations but not discard your investments or replicate architectural patterns that can inhibit the data and insight that organizations need to modernize. Although going to the cloud has a great deal to offer, doing so also might lock organizations into a situation that is not dissimilar from the status quo approach. Also, the cloud can further exacerbate the time that it takes to access data, and work in opposition to closing the data latency gap that is required to deliver business-critical functions.

1.5 What can you do?

Your organization can address the challenges of your current architecture to better use data, close the data latency gap, and embed data and insight into business processes where suitable.

You can access data at its source; therefore, critical data-driven decisions can be made before an interaction or transaction completes and your customer abandons their interaction with your organization. Your architectural strategy can ensure access to data across platforms and multiple clouds, whether the data is structured or unstructured.

1.6 Resolving the data latency gap through data virtualization

Data virtualization is emerging as an exciting, cost-effective, substitute for and augmentation to traditional data collection (incremental copy, data movement, and ETL). With data virtualization, you can access data where it originates to reduce the time and resources that are used to combine data from multiple systems. Less time and resources can translate into savings. The use of data virtualization technology can support greater flexibility and agility, which is core to digital transformation.

1.7 DVM for z/OS

DVM for z/OS provides virtual, integrated views of data that is on IBM Z. It also enables users and applications to read/write access to IBM Z data in place, without moving, replicating, or transforming data. It also performs these tasks with minimal extra processing costs. By unlocking IBM Z data using popular, industry-standard APIs, DVM for z/OS can save you time and money.

Developers can readily combine IBM Z data with other enterprise data sources to gain real-time insight, accelerate deployment of traditional mainframe and new web and mobile applications, modernize the enterprise, and take advantage of today's API economy.

DVM for z/OS also supports data access and movement. As with many technologies, the best approach depends upon the specific needs. DVM for z/OS can be considered part of a larger holistic approach to data delivery.

1.8 IBM Cloud Pak for Data

IBM Cloud Pak for Data is a fully integrated data and AI platform that modernizes how businesses collect, organize, and analyze data and infuse AI throughout their organizations. Built on Red Hat OpenShift Container Platform, IBM Cloud® Pak for Data integrates market-leading IBM Watson® AI technology with IBM Hybrid Data Management Platform, data ops, and governance and business analytics technologies. Together, these capabilities provide the information architecture for AI that meets your ever-changing enterprise needs.

Deployable in just hours and easily extendable with a growing array of IBM and third-party microservices, IBM Cloud Pak® for Data runs across any cloud, which enables organizations to more easily integrate their analytics and applications to speed innovation.

IBM Cloud Pak for Data lowers your total cost of ownership, accelerates innovation that is based on open-source technologies, and fully supports multi-cloud environments, such as Amazon Web Services (AWS), Azure, Google Cloud, IBM Cloud, and private clouds.

When the data on which you are building your machine learning models originates in relational IBM Z data sources (Db2 for z/OS) and traditional non-relational IBM Z data sources (such as IMS, IBM MQ, and VSAM), IBM Cloud Pak for Data provides integrated connectivity through DVM for z/OS. This connectivity significantly simplifies the development of analytics applications that are driven from IBM Z data, which allows developers and data scientists to readily access complex data structures by way of SQL.

IBM Cloud Pak for Data provides a data virtualization service with integration to DVM for z/OS. This service provides the following functions:

- ▶ Facilitates connectivity and access to data sources that are configured by DVM for z/OS
- ▶ Supports business rules and policies that can be applied at the exposure point within Cloud Pak for Data for downstream traditional and modern mainframe applications
- ▶ Provides lineage for downstream applications that access, transform, and deliver data
- ▶ Administers and provisions data virtualization for non-IBM Z data sources

1.9 Unlock enterprise data for virtually any application

Hybrid cloud applications incorporate on-premises and on-cloud services and use all data across the enterprise. Modern applications are interconnected, interact through APIs, and enable customers and enterprises to digitally run processes quickly. These applications need agile read/write access to IBM Z data, both relational and non-relational, in an online environment.

Applications that use traditional, scheduled batch programs to update transactional data can be refactored into modern applications that can access and update IBM Z data by using modern APIs that are supported by DVM for z/OS. Because DVM for z/OS runs almost exclusively on readily available IBM Z Integrated Information Processors (zIIP), it does not use general processor capacity. It also might significantly reduce mainframe costs that are associated with ETL.

DVM for z/OS can virtualize established data sources, such as virtual storage access method (VSAM), adaptable database system (ADABAS), IBM IMS/ESA® Database Manager, IBM Db2 for z/OS, and IBM System Management Facility (SMF). Its ability to federate these sources with virtually any other data brings the power of IBM Z essentially to any application, mobile, analytic, or cloud. It does so with minimal extra processing costs and without the need for IBM Z skills or more coding.

Reducing complexity for accessing data applications implies less time to implement new engagement applications while also accessing real-time data. This means fewer unique skill sets are required to access complex established data structures. Complex application programming can now be done with simple SQL and NoSQL access. Updating or building modern applications by using IBM Z data can produce elastic, interconnected, and more secure applications to deliver a competitive advantage.

DVM for z/OS is optimized for the hardware it runs on and takes advantage of the new instruction sets available with each new IBM Z hardware platform. As IBM Z offers new hardware capabilities, DVM for z/OS inherits those advantages.

1.10 Why and when should you consider DVM for z/OS

Any organization with data on IBM Z, whatever the data type, can benefit from DVM for z/OS. DVM for z/OS allows you to:

- ▶ Combine and integrate non-relational and relational data, integrate IBM Z and non-IBM Z data, and incorporate unstructured data with structured data.
- ▶ Provide SQL access to non-relational data.
- ▶ Modernize mainframe applications and enhance non-mainframe applications with mainframe data.

Your alternatives are limited only by your imagination.

When your valuable data originates on IBM Z and moving it off platform exposes you to cost, latency, or risk, the use of the processing on IBM Z can be your best alternative. IBM DVM for z/OS helps you access your data where it originates.



Architecture and implementation

IBM Data Virtualization Manager for z/OS (DVM for z/OS) is the only z-resident data virtualization technology that takes full advantage of the IBM Z platform for secure, scalable access to accurate, on-time data at a moment's notice. An enterprise chooses to incorporate this technology in their data processing methodology for the following reasons:

- ▶ Real-time query: Provides real-time access to in-place data
- ▶ Optimization: Speeds up high-volume data retrieval and uses zIIP optimization
- ▶ Modernization: Provides a data service layer for your API economy, such as REST, to access native data on IBM Z without application changes

DVM for z/OS is different from other so-called virtualization or replication tools because specialized training or expert database skills are not needed. You can work with data structures that use virtual tables and virtual views that are familiar to you, and provide users and applications read/write access to IBM z/OS and enterprise data sources in real time.

DVM for z/OS enables organizations to access their mainframe data through virtual, integrated views without the need to move, replicate, or transform it, which saves time and expense.

This chapter describes the fundamental architecture of the IBM Data Virtualization Manager for z/OS (DVM for z/OS or just DVM) technology and includes the following topics:

- ▶ 2.1, "Reference architecture for DVM for z/OS" on page 10
- ▶ 2.2, "Technical components" on page 10
- ▶ 2.3, "SQL engine and query optimization" on page 12
- ▶ 2.4, "Metadata repository" on page 14
- ▶ 2.5, "Parallel processing through MapReduce" on page 16
- ▶ 2.6, "z/OS resident optimization" on page 17
- ▶ 2.7, "zIIP eligibility and data compression" on page 18
- ▶ 2.8, "DVM Studio" on page 19
- ▶ 2.9, "Integrated DRDA Facility" on page 21
- ▶ 2.10, "DVM endpoint connections" on page 23
- ▶ 2.11, "Summary" on page 28

2.1 Reference architecture for DVM for z/OS

DVM for z/OS offers organizations the potential to modernize the way they develop applications and simplify access to an extensive number of traditional non-relational IBM Z data sources. It incorporates various interface approaches for accessing data that uses IBM Z Integrated Information processors (zIIP) to redirect up to 99% of workloads. The amount of zIIP offload activity varies with different types of workloads, depending on the data source, access method, and effective SQL operations in use.

The DVM server supports IBM-supported hardware ranging from z196 to the latest IBM models, running IBM z/OS v1.13 or later.

The technology supports traditional database applications, such as Db2 for z/OS, IMS, IDMS, and ADABAS. Also, typical mainframe file systems, such as sequential files, ZFS, VSAM files, log-stream, and SMF also can be accessed.

DVM reduces overall mainframe processing usage and costs by redirecting processing that is otherwise meant for GPPs to zIIPs. DVM provides comprehensive, consumable data that can be readily accessible by any application or business intelligence tools to address consumer demand and stay ahead of the competition.

DVM for z/OS includes the following key features:

- ▶ Provides a layer of abstraction that shields developers from unique data implementation
- ▶ Virtualizes IBM Z and non-IBM Z data sources in place in real time
- ▶ Supports modern APIs, such as JDBC, ODBC, SOAP, and REST (requires z/OS Connect Enterprise Edition)
- ▶ Supports MapReduce for faster data access
- ▶ Offloads up to 99% of General Processing to lower-cost zIIP specialty engines
- ▶ Supports Db2 and IMS Direct for efficient large data retrievals
- ▶ Supports data encryption

2.2 Technical components

As with CICS, Db2, and IMS, DVM for z/OS runs as a started task that stays running to service client requests. As a resident, it acts as a modern data that uses modern APIs. It also provides virtual, integrated views of data that is on IBM Z.

Users and applications have read/write access to IBM Z data in place, without moving or replicate the data. It performs these tasks at high speed, without more processing costs. By unlocking IBM Z data that uses popular, industry-standard APIs, you save time and money.

Developers can readily combine IBM Z data with other enterprise data sources to gain real-time insight, and accelerate deployment of traditional mainframe, and modern mobile applications.

Figure 2-1 provides a logical view of the DVM architecture. Data consumers and providers that interact with DVM are listed on the left side of the figure. Although not an exhaustive representation of all the types of applications, tools, or repositories that need access to data in a relational format, Figure 2-1 shows the demand for data across all areas of an enterprise.

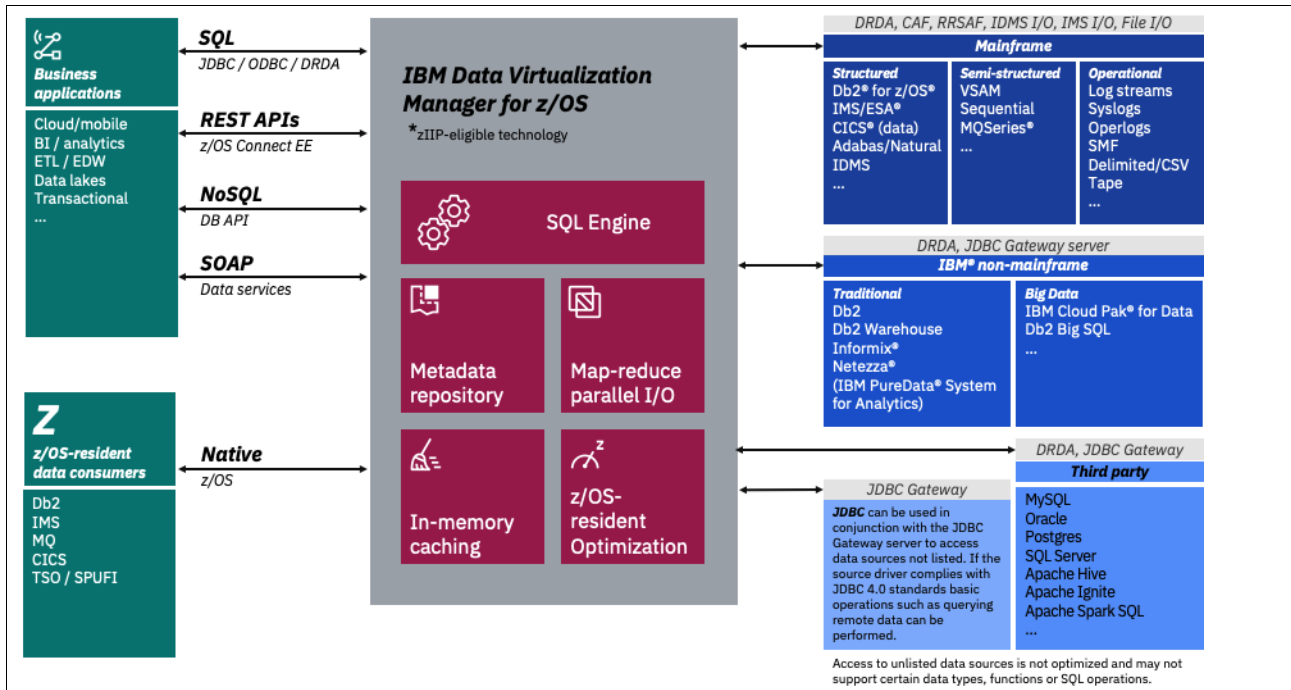


Figure 2-1 DBM for z/OS architecture

These data consumers connect to the DVM server through APIs that use JDBC, ODBC, DRDA drivers over available network ports. However, the more common web or mobile interface applications connect by using IBM z/OS Connect Enterprise Edition (zCEE) that use RESTful APIs.

The right side of Figure 2-1 represents common z/OS and non-z/OS data sources, both structured and semi-structured. After data sets are virtualized, the DVM server supports the joining of supported data sources as referenced in the IBM documentation for DVM for z/OS.

2.3 SQL engine and query optimization

The DVM server, as shown in Figure 2-1 on page 11, enables applications to access virtualized enterprise data in real time by using ANSI 92 / 99 SQL statements.

2.3.1 Query processing by SQL engine

DVM enables the user to define any content as a data source to create a virtual table to map to what is stored within the metadata of the DVM server. When a user runs a database call, the SQL engine reads the mapping and builds a table structure in memory.

As a virtual table or virtual view is materialized, the SQL engine analyzes the best access path to fetch data by using proprietary parallelism algorithms that are built into the product. The SQL engine then applies filters and functions against retrieved data for the SQL statement issued.

When a query is received by the SQL engine, the SQL statement is parsed as part of PREPARE processing. During the parsing operation, individual subtables that are referenced in the query are identified and associated virtual table definitions are consulted.

For data sources with indexes, the SQL is examined to see whether index key fields are used in the predicate. The use of key fields can reduce the amount of data that is fetched and can be used to optimize joining data with other virtual tables. If MapReduce is used, retrieval of each data is apportioned out to separate threads, and when possible, WHERE predicates are pushed down to MapReduce threads to limit the total amount of data that is fetched.

2.3.2 WHERE predicate PUSHDOWN

Predicate PUSHDOWN is a common federation technique that uses filtering at a data source. Specific parts of SQL queries (the filtering predicates) can be “pushed” to where the data lives.

Predicate PUSHDOWN is an optimization technique that can drastically reduce query and processing time by filtering data earlier within data access. Depending on the processing framework, predicate PUSHDOWNs can optimize your query by performing tasks (such as filtering data before it is transferred over the network) as part of pre-load into memory, or by skipping the READ of entire files or chunks of files.

The DVM server requires statistics from the underlying data source to effectively perform predicate PUSHDOWN.

2.3.3 Join processing

When processing joins between two tables, the DVM SQL engine uses previous executions to identify the smaller table and then loads the smaller one into memory, which matches the entries from the larger table. Approaching join operations in this manner helps to optimize the use of memory and speeds processing.

2.3.4 Referential integrity

Referential integrity (RI) is outside the scope of the DVMServer. Non-relational data sources, such as VSAM, do not have a built-in RI layer. RI can be accomplished within the application by validating records before they are written or updated.

For deletes, the application must address data integrity and references by making sure other identified records exist. Otherwise, databases can support referential integrity between tables.

Db2 for z/OS is designed to manage its own referential integrity by using DDL statements to define primary and foreign keys and rules. When these tables are virtualized, the DVM server sends DML statements to the Db2 subsystem for processing when the Db2 Direct-access method is not used. Db2 referential integrity rules are maintained for Inserts, Updates, and Deletes.

2.3.5 Virtual Parallel Data

Virtual Parallel Data (VPD) is a scalability feature that is built into the DVM runtime. VPD simulates a data cache for the DVM server to access any file or database once so that applications can share or use the file or database simultaneously across highly concurrent client tasks.

Figure 2-2 shows the I/O relationship between the DVM SQL engine and cached data. VPD is well suited for data sets that are frequently accessed, but are not dynamically changing throughout the day.

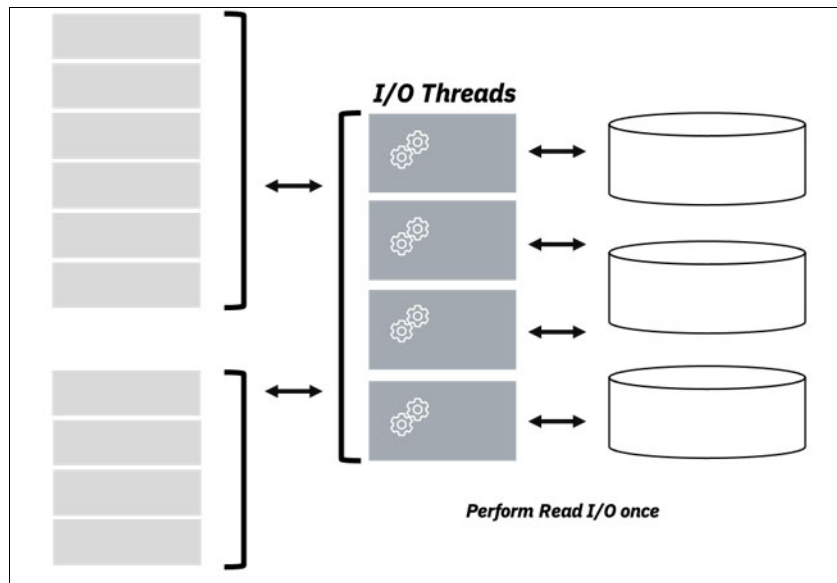


Figure 2-2 DVM VPD

VPD also allows multiple requests to run with asymmetrical parallelism, separately tuning the number of I/O threads and the number of client or SQL engine threads. When I/O is limited to a single task, VPD allows parallelism in SQL as listed in Table 2-1 on page 14.

Table 2-1 Asymmetrical parallelism

Test environment	Number of SQL threads	I/O threads	Elapsed time
MR=N	1	1	25.234
MR=Y	1	6	24.488
VPD with MRCC	6	1	10.821
MRCC	6	6	8.954

These results demonstrate how a VPD data cache can greatly improve execution time for SQL over increasing concurrent usage. Instantiating the VPD enables a readily accessible cache of data and avoids subsequent and concurrent access requests to disk. This reduces the I/O processing for the system, improves latency, and speeds up the query performance. VPD cache can be refreshed daily or as scheduled.

The first query against the VPD (see Example 2-1) creates the group and I/O immediately populates the cache. An initial I/O is performed once and buffered in one or more large, 64-bit memory objects. If multiple groups comprise a VPD, they all share the buffered results. Each VPD request can specify its own degree of parallelism by using the MapReduceClient (MRC) and MapReduceClientCount (MRCC) configuration parameters with the JDBC driver.

Example 2-1 Query against the VPD

```
SQL:          SELECT * FROM ED_ALL_COLS_PS_CMPSTRP6
Data set:     RSTZST.HLO.DBHLOEXM.UNLOAD.CMPSTRIP.SYSREC
Rows:        14,000,000Bytes: 1,484,000,000
```

2.3.6 Flatten arrays

It is common to encounter arrays within a data record (for example, OCCURS clause in COBOL copybook). The DVM server provides options for how arrays can be processed. Arrays can be flattened into a fixed-length row or normalized into a main or subtable virtualization table pair.

VSAM data sets can include multiple occurrences of a value. For example, a single record within a VSAM file that describes billing information can contain the last years' worth of payments. A relational structure is represented as two tables within a normalized design or a single table with multiple records that represent a single occurrence of each record for one occurrence within the VSAM file.

2.4 Metadata repository

The DVM server maintains a metadata repository, which is a collection of information that describes the data that can be accessed by the DVM server. The DVM server uses all that information to connect to the data source, convert SQL calls into the original data sources access method, and return results to the application.

To access data, you must first virtualize the data as a table by defining a virtual table as a container for all the attributes associated with the data source. For example, when an SQL statement is issued against a VSAM record.

The DVM server retrieves the fields from the metadata repository that represent the data set. The server then converts the SQL into native VSAM I/O calls and returns the data to the requester. This approach allows data to stay in place and enables the structured, unstructured mainframe databases and files to be represented in a relational format. The DVM server establishes an implicit table schema to file formatting.

The DVM server processes data in real time, which avoids creating copies of data and supports transactions that write back to the original data sources, whether online or offline, on-premises, or in the cloud. Data is not cached, nor does DVM require a specific API schema; therefore, organizations have flexibility regarding API naming conventions.

2.4.1 DVM catalog tables

All metadata is stored within the DVM server. Metadata is also known as *DVM catalog tables*. Applications that use DVM's SQL engine through JDBC or ODBC can use SQL to view this data. You can also transfer (or access) DVM metadata from COBOL, PL/I, MFS maps, and stored procedures into a library or lifecycle management system for optimized use, as shown in Figure 2-3.

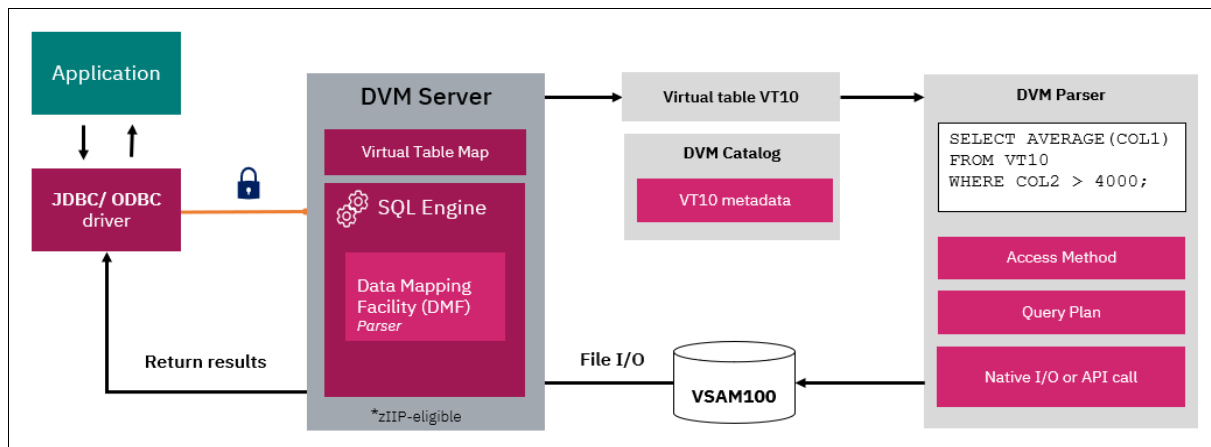


Figure 2-3 DVM catalog tables

The following metadata objects are stored as members in the map data set:

- ▶ Virtual tables (VSAM, IMS, Db2, Adabas, IDMS, sequential, and so on)
- ▶ Virtual views
- ▶ Application artifacts (IMS DBD, IMS PSB, COBOL copybooks, PL/I structure, and assembler DSECTs)
- ▶ Stored Procedures
- ▶ Remote target systems
- ▶ Virtual source Libraries
- ▶ Web services

The metadata repository is stored as members within IBM Z Partitioned data sets (PDSs). Metadata maps are loaded from data set members and cached in memory each time the DVM data server is started.

Maps for virtual tables are automatically added to auto-generated system catalog tables, which can be queried through standard SQL. DVM server metadata is stored in catalog tables, which are built and loaded when the address spaces are started and when a map refresh operation is performed. These maps are stored as PDS members in the servers map data set. When data maps are created by using DVM Studio or through a batch import utility, each new data map is imported into its respective mapped data set and refreshed in the online catalog cache.

The following DVM catalog tables can be queried and are valuable for understanding the relationships between entities, which are useful in joining tables:

- ▶ SQLENG.TABLES
- ▶ SQLENG.COLUMNS
- ▶ SQLENG.COLUMPRIVS
- ▶ SQLENG.PRIMARYKEYS
- ▶ SQLENG.PROCEDURES
- ▶ SQLENG.STATISTICS
- ▶ SQLENG.TABLEPRIVS
- ▶ SQLENG.FOREIGNKEYS
- ▶ SQLENG.SPECIALCOLS
- ▶ SQLENG.TABLES

2.5 Parallel processing through MapReduce

The DVM server optimizes performance through a multi-threaded z/OS-based runtime engine that uses parallel I/O and MapReduce. With parallel I/O, threads are running in parallel to simultaneously fetch data from the data source and return the result to the client, as shown in Figure 2-4. Each interaction that is initiated by a distributed data consumer or application runs under the control of a separate z/OS thread.

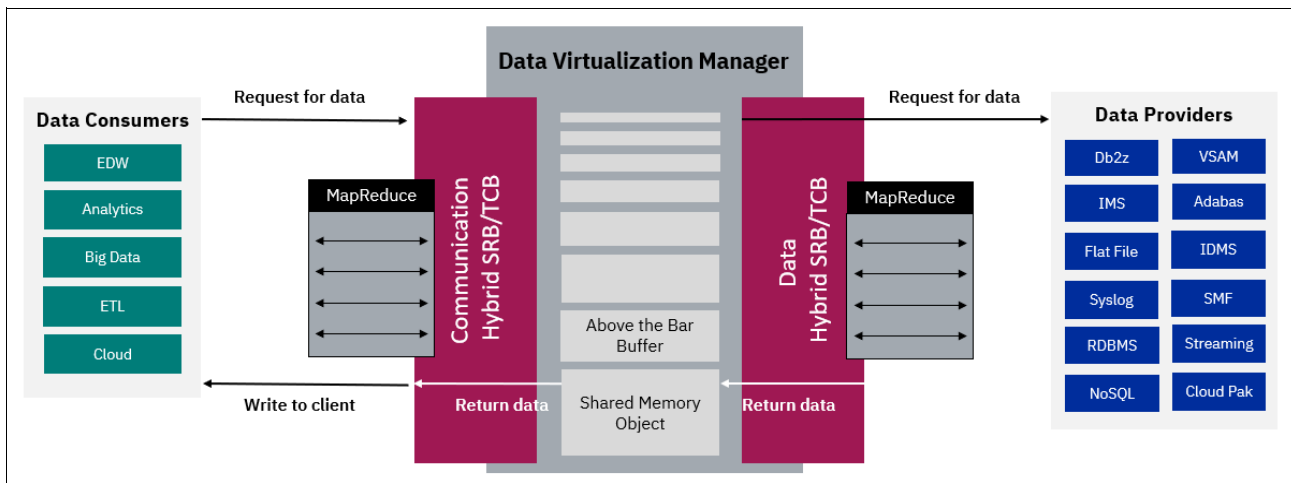


Figure 2-4 Parallel IO and MapReduce with DVM

MapReduce significantly reduces the query elapsed time by splitting queries into multiple threads, which run in parallel and aggregate the data into a single result set. DVM reduces the cost and complexity of data movement through data integration with Hadoop, EDW, Cloud, and other distributed sources through SQL calls to join large disparate data sources for simplified access.

The DVM server uses PREDICATE PUSHDOWN and index keys where possible. When joins are used, DVM pushes down the appropriate filters that apply separately to each of the disparate data sources that are involved in the join.

2.6 z/OS resident optimization

DVM is a resident of z/OS. Because it was written in IBM Enterprise Metal C for z/OS, it can be compiled to optimize the latest version of the IBM Z hardware on which it runs.

With each new generation of IBM Z hardware, DVM uses instruction-level performance improvements automatically. Upon starting a DVM task, DVM loads the optimized modules for the generation of the IBM Z processor in service. DVM is purposely designed to reduce Supervisor Calls (SVCs) and provides up to 99% zIIP-eligibility.

Although not Java-based, the DVM server runs almost entirely in Enclave SRB mode, where resources that are used to process workloads can be accounted to the transaction, rather than to the address space in which the transaction runs.

DVM can provide performance benefits in reading large data sets by circumventing Db2 and IMS Subsystems. DVM also can access and read the underlying VSAM data sets that are used by IMS and Db2 for z/OS. This ability reduces the total cost of ownership for the solution and improves performance for bulk-load operations. As a Z-resident software solution, the DVM server provides the following significant optimizations:

- ▶ Dynamic parallelism against large z/OS data sets: logical partitioning of Flat, VSAM IMS, Db2 Log Streams, Adabas, IDMS, and so on.
- ▶ Use of:
 - Pageable Large Frames for DAT reduction
 - Single Instruction Multiple Data SIMD
 - Optimized AIOCB TCP/IP APIs
 - Shared Memory Objects for inter-process communication, which significantly reduces data movement
- ▶ High-Speed Tracing facility that is built on Data In Virtual (DIV) services.
- ▶ An ACEE cache that is tied into ENF 71, which avoids accessing the RACF database and easily keeps the cache in sync.
- ▶ Unique use of zEDC for improvements in network I/O compression.
- ▶ Pre-built infrastructure for DVM to CICS interfaces, which eliminating five out of the six API calls that are required to use EXCI.
- ▶ Automatically “compiled” DVM REXX to run REXX as zIIP-eligible and in cross-memory mode. However, HLLs provide a minimum opportunity for zIIP eligibility. DVM REXX is representative of this poor zIIP eligibility.

2.7 zIIP eligibility and data compression

The IBM Z Systems-Integrated Information Processor (zIIP), as shown in Figure 2-5, is a dedicated specialty processor that operates asynchronously with the General Purpose Processor (GPP) on a mainframe system for specific workloads. zIIP processors also help to manage containers and hybrid Cloud interfaces, and facilitate system recovery, assist with several types of analytics, and system monitoring.

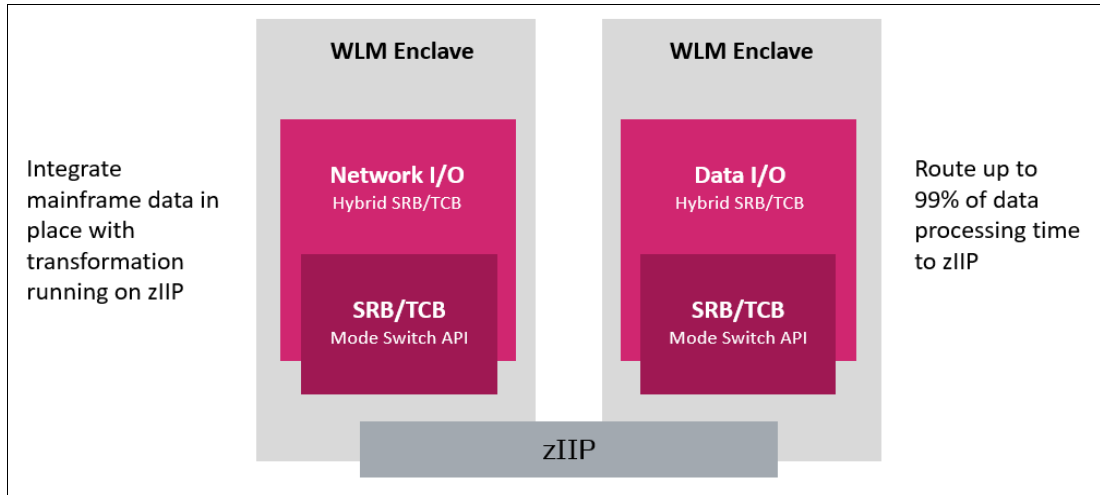


Figure 2-5 System Z Integrated Information Processor

zIIP specialty engines deliver higher performance and lower cost than general purpose processors (GPPs), which are not typically fully used. zIIPs can handle specialized workloads, such as large data queries over Db2, Java, and Linux, to redirect processing from GPPs.

The DVM server automatically detects the presence of zIIP engines and transitions qualifying workloads from the GPP to a zIIP engine. The result is reduced mainframe processing usage and cost.

Also, the DVM server takes full advantage of IBM's zEnterprise® Data Compression (zEDC) in z/OS 2.1, along with the Integrated Accelerator for zEnterprise Data Compression. In recent models of IBM Z hardware, data compression is performed by the integrated compression co-processor, which allows sharing large amounts of compressed data with the DVM server. This ability reduces data transfer latency, improves CPU usage, and saves disk space.

2.8 DVM Studio

DVM Studio is an Eclipse-based user interface that is included with the DVM server. DVM Studio can be installed as a stand-alone or add-in component to an eclipse framework. It allows experienced and novice mainframe developers and administrators to readily define virtual tables from non-relational data. It also allows developers to generate code snippets in a broad range of APIs and interfaces.

Figure 2-6 shows the client connection between DVM Studio and the DVM server.

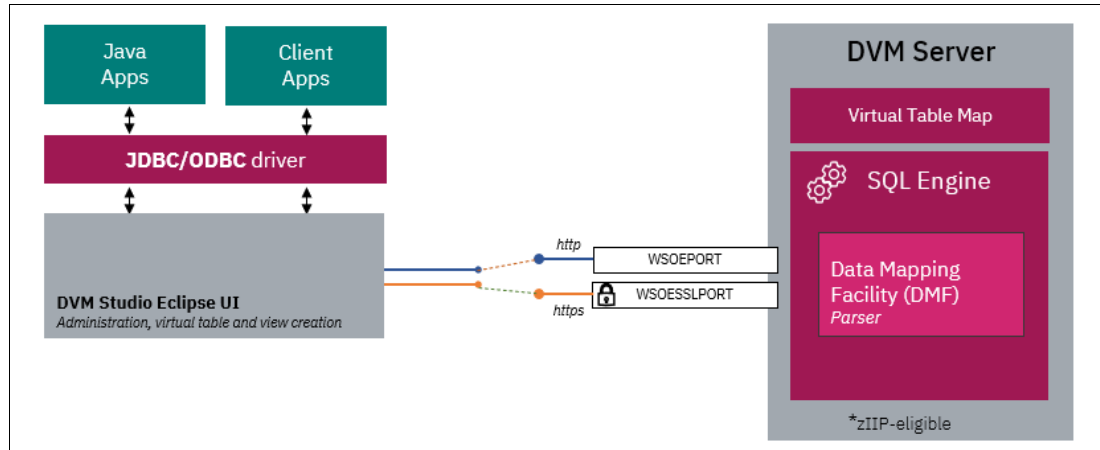


Figure 2-6 Eclipse-based DVM Studio

DVM Studio is distributed with a Java Database Connectivity (JDBC) JVM 1.4 or higher. DVM Studio can connect to the DVM server and use the JDBC driver to connect to the server or by using an HTTP call.

DVM Studio also offers an application development environment that simplifies application development through an automated code generation utility and supports modern programming languages. It provides views, wizards, and editors with which you can build, develop, and transform mainframe applications into web services and components.

System programmers and DBAs typically collaborate to identify data sets on the mainframe and create schema-related artifacts that describe underlying data as a first step to the use of the Studio. After the artifacts are available and suitably describe the associated descriptors for data types, column descriptions, and so on, they can be easily mapped and provisioned for use by DVM Studio.

After underlying data sets, such as databases, VSAM files, system files, and tapes, are discoverable through DVM Studio, DBAs, and developers can define virtual tables and views for more productive use. Developers can use DVM Studio as a complementary IDE-like framework to their primary source control environments by creating and publishing a web service or generating development code by using SQL.

DVM Studio features a built-in code generator capability that is perfect for creating code snippets in your favorite programming language. The code generator produces code snippets in Python, R Studio, Spark, Scala, Java, Jupyter Notebooks, and more.

Figure 2-7 shows the workflow for a DBA or Developer in DVM Studio.

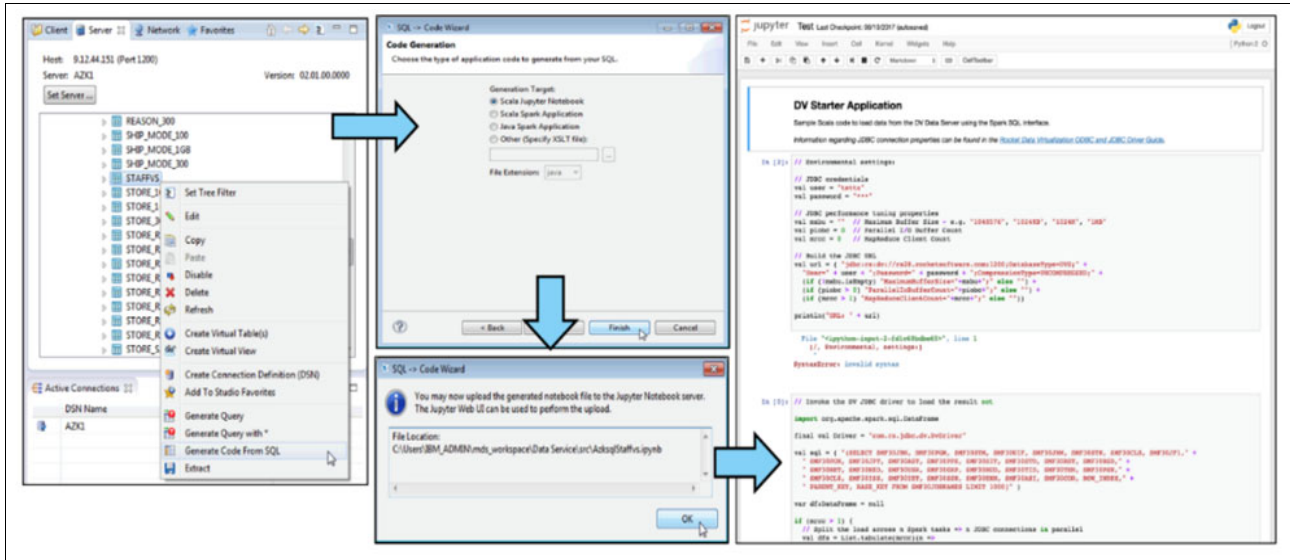


Figure 2-7 Publishing a web service by using DVM Studio

You can generate and publish web services directly from DVM Studio, or automatically generate Db2 user-defined table functions (UDTFs) as part of a Virtual Data Facility. This technique uses the Db2 for z/OS subsystem as an information hub and redirects applications through Db2 for access to other mainframe and non-mainframe data sources.

IBM's Db2 distributed databases offer a built-in data federation engine that uses nicknames or remote tables in a similar manner. Figure 2-8 shows the Virtual Table wizard and SQL editor.

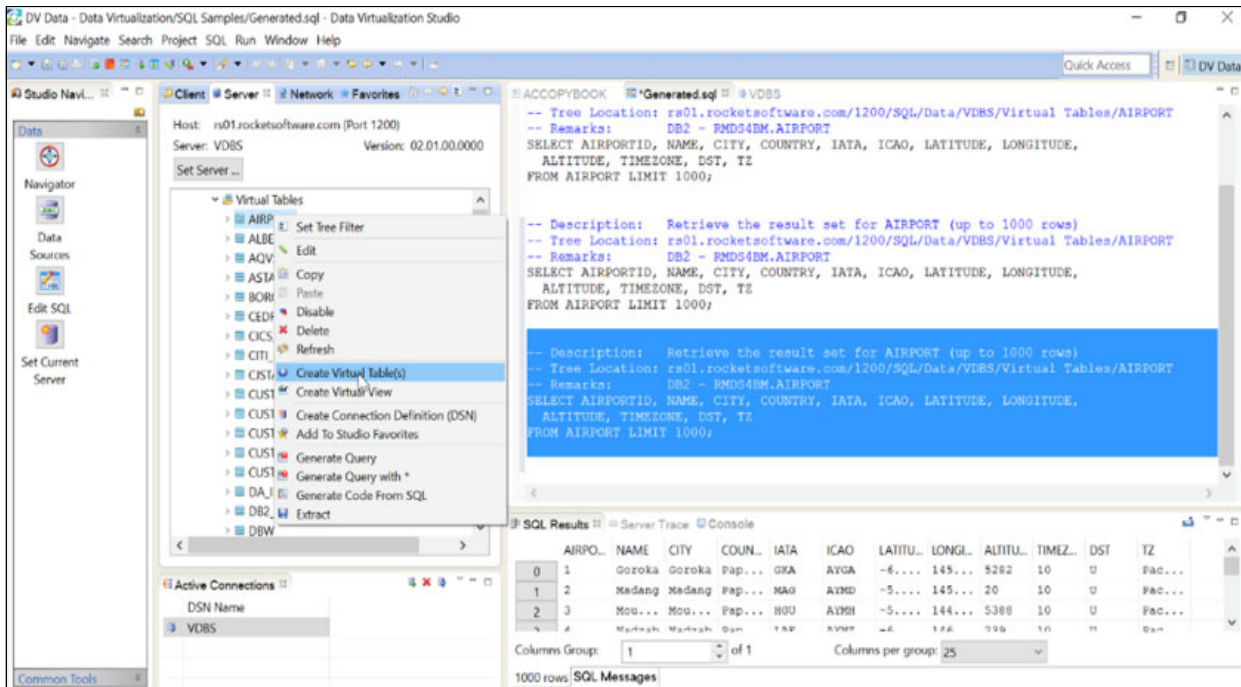


Figure 2-8 DVM Studio Virtual Table wizard and SQL editor

DVM Studio is used to define virtual objects, such as virtual source libraries, virtual tables, virtual collections, and virtual views on the host DVM server. Administrators can easily create virtual tables and views and apply predicates.

Consider the following points:

- ▶ Virtual source libraries exist on the mainframe and point to the information (metadata) that is required to virtualize the source data.
- ▶ Virtual tables provide a physical mapping to the data that you want to access from the data source. After the virtual table is defined, use it to generate and run SQL. The resulting SQL is used to read and extract the mapped data from the mainframe.
- ▶ Virtual views can be defined across one or more virtual tables, from which you can generate SQL queries. A virtual view contains the columns from more than one database object within or across homogeneous and heterogeneous data sources.

2.9 Integrated DRDA Facility

The Integrated DRDA Facility (IDF) enables the DVM server to become a DRDA Application Server (AS). The DVM server can be connected to another DVM server by using peer-to-peer communication and also a Db2 subsystem as an AS. This capability enables a Db2 subsystem to access virtual tables and views by using three-part table names.

The new IDF feature allows a DVM server to participate as an endpoint in this kind of multi-homed configuration. Whenever SQL is run in Db2 and a three-part table name designates an IDF endpoint, the SQL is sent to the DVM server for execution. Results are returned to Db2z as though it were a local table.

2.9.1 Peer-to-peer configuration

IDF connects a DVM server to another DVM server to offer seamless access throughout your data ecosystem. IDF enables access to a peer DVM server and a shared view of locally virtualized data objects, such as virtual tables and views.

IDF allows data sources to be accessed by local clients running on a specific z/OS LPAR or by participating in a sysplex environment on a different z/OS LPAR or sysplex.

Imagine your organization has a data center in North America and another in Australia, each running IMS, and each having its own inventory databases. To reconcile their inventory systems, it requires a manual process.

In another scenario, a data source exists on LPAR PRD2, but it is not on a shared disk or available for access on all your LPARS. Most of the users and applications are running on LPAR PRD1 and include requirements to access the data because they want to join with Db2, which is running on PRD1. Users on PRD1 or PRD2 can now retrieve data from Db2 and join it with virtualized data on PRD2 by using IDF.

By installing DVM on each LPAR and connecting to each server as shown in Figure 2-9, each data center can view each other's virtualized data by right-clicking any of those tables.

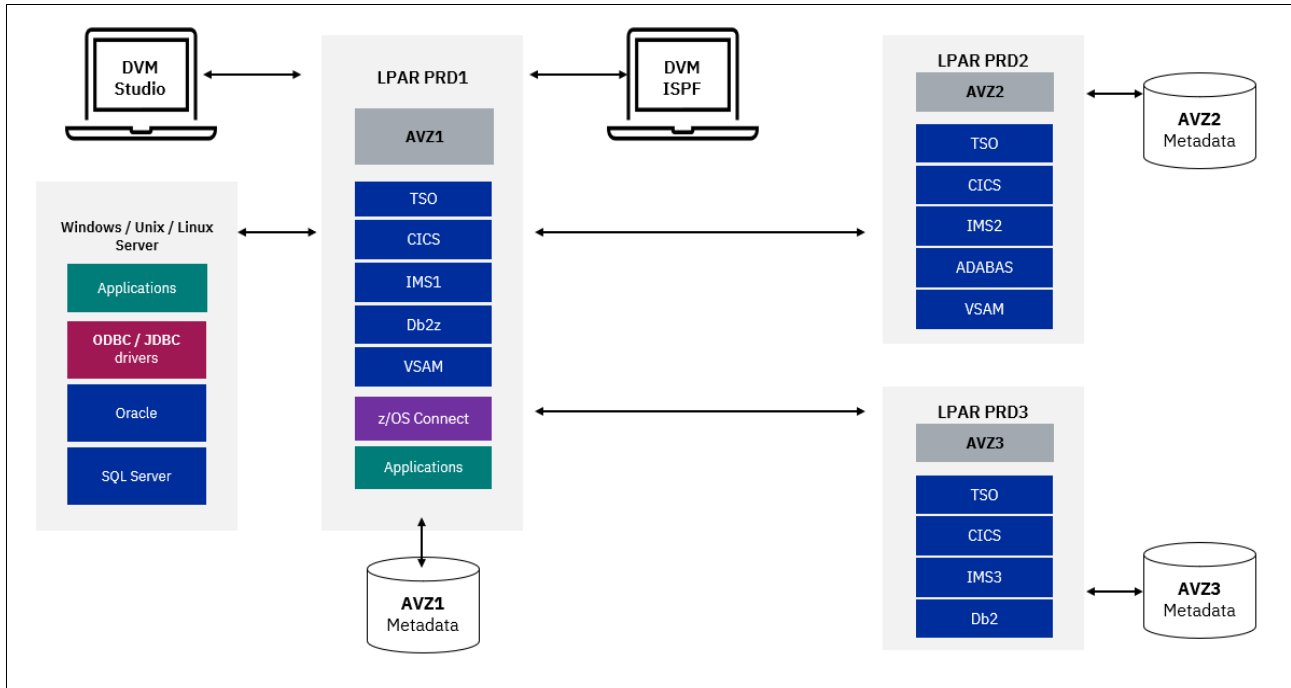


Figure 2-9 DVM Studio

DVM adds it to the metadata on their server. IDF peer-to-peer makes it appear as though all of the data is local and available no matter where it is stored. Instead, you can run SQL to perform join operations, rather than copying the data and writing Cobol or PL/I programs to do the reconciliation.

2.9.2 Db2 Information Hub

You can also use DVM and IDF as a Db2 data hub by connecting Db2 to the DVM server by using a Db2 DRDA connection, where Db2 becomes the application requester (AR) and the DVM server becomes the application server (AS)

After the Db2 subsystems are configured for a DRDA connection, DVM supports a full-featured implementation in which your Db2 applications have transactional access to all participating data sources. IDF uses a three-part naming convention to map to the remote data objects, whereby it identifies data based on location.

Within the same application, you can simultaneously update many different data sources without needing to direct applications individually to each host.

2.10 DVM endpoint connections

DVM provides a highly flexible connection framework that offers various client access methods to the DVM server, as shown in Figure 2-10.

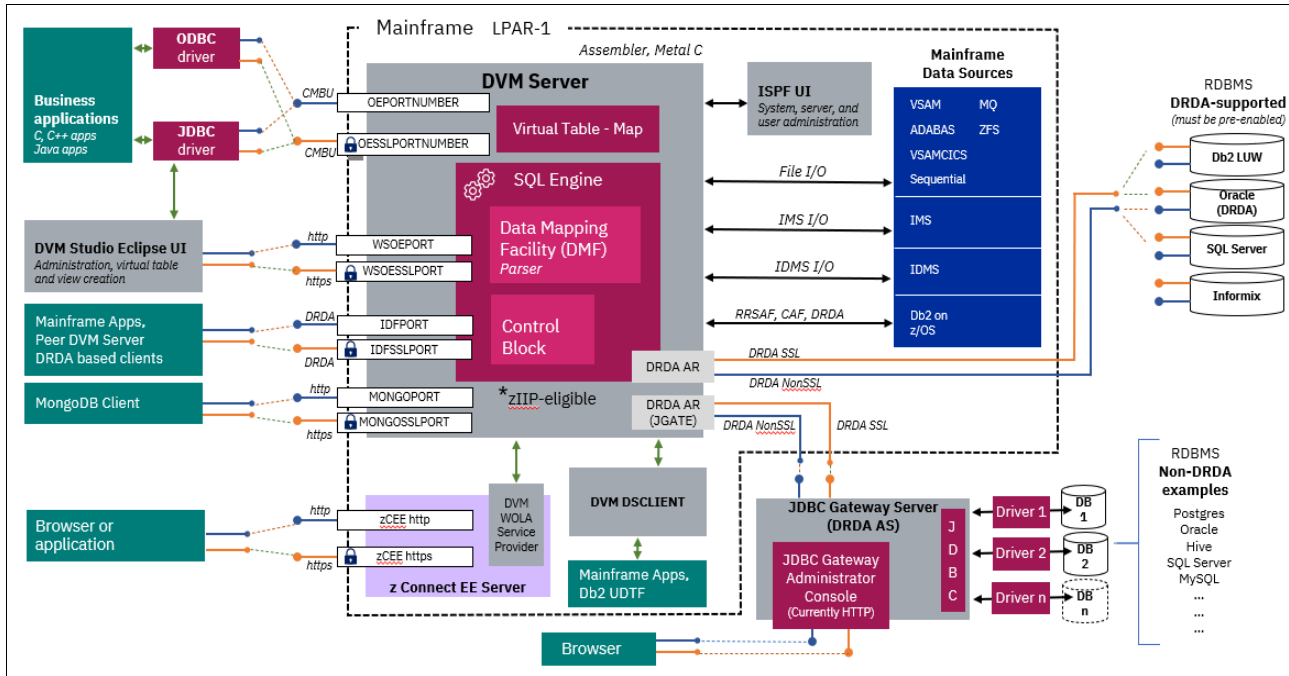


Figure 2-10 DVM and endpoint technical view

2.10.1 Drivers

Any C, C++, or Java application can access mainframe data with the DVM SQL engine and its ODBC and JDBC drivers. The communication between the driver and the DVM server is a proprietary communication buffer protocol (CMBU) or communication buffer.

Two ports are dedicated to handling these requests with two different levels of security. The orange connector depicts an SSL-enabled connection method to a port that is configured with a TLS encryption protocol; the blue connector depicts a non-SSL connection method to an unencrypted port, as shown in Figure 2-11 on page 24. IBM recommends the use of secure ports. These ODBC and JDBC drivers also are included in the IBM Cloud Pack for Data as part of the built-in connector framework that is used to define connections with remote data sources.

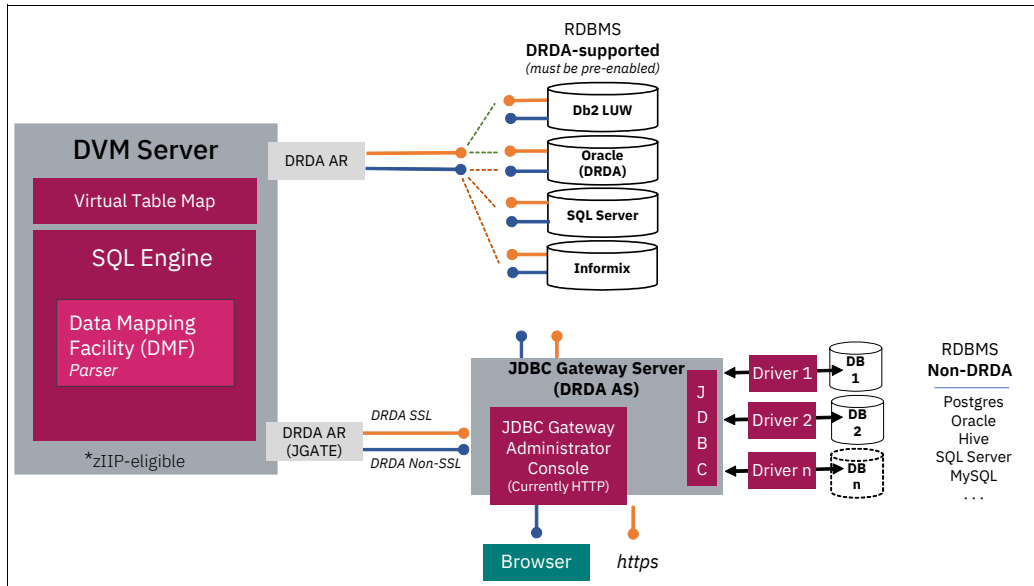


Figure 2-11 DVM communication buffer protocol

The ODBC driver can be used to connect many C and C++ applications to the DVM server. Many programming languages are available for use with the ODBC protocol, such as Python. Java applications can use the JDBC driver to connect to the DVM server. Also, specific languages, such as Python, can make use of our code to connect and run a request.

2.10.2 DVM Parser and Data Mapping Facility

The DVM Parser is primarily used to parse source data structure components that are used to define data access. Such components can be a copybook in a COBOL or PL/I programs that define a record layout or the source code that defines a DBD and PSB that is used for IMS structures. The Data Mapping Facility is the component of DVM that is responsible for creating a virtual table and virtual views, and defining the metadata (XML document) that is used for mapping access to the physical data.

The parser is a component of the DVM server and can be started through Job Control Language (JCL) batch jobs. A version of the parser also is distributed with DVM Studio installation (mainly targeting Windows, but potentially UNIX, Linux, and Mac platforms).

DVM Studio distributed version of the Parser typically is used to generate an internal XML representation of the source document. It then uses this information through its various wizards to ultimately generate the DVM metadata for the scenario in question (that is, the Cobol copybook layout is transformed into DVM metadata format).

DVM Studio uses a combination of the DMF and the parser to build the definition of virtual tables and views that are used by DVM, as shown in Figure 2-12.

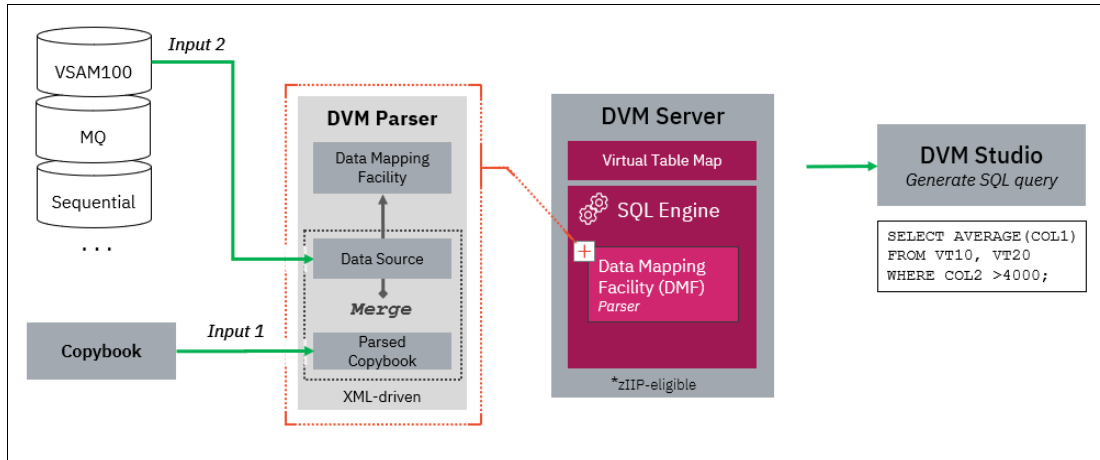


Figure 2-12 DVM parser

The parser is used to read the Source metadata and transform it into the DVM metadata format. Some of the source metadata types that the DVM server can accept are listed in Table 2-2.

Table 2-2 Artifacts that is required to create virtual tables by data source

Data source	Access method	Source metadata	Physical file
Sequential VSAM		COBOL copybook	PDS data set
IBM DB2® for z/OS	Pass-thru Direct UDTF		
IBM DB	DBCTL ODBA Direct	DBD, PSB	
ADABAS		ADABAS FDT	
SMF		HTML DSECT	Sequential
Syslog		DSECT	Sequential
Slog stream		DSECDT	Sequential

2.10.3 DS-Client API interface

The DS-Client API interface provides an alternative method to connect applications to the DVM server, in the case where customers are not running Db2 and require an application interface to DVM for z/OS.

The DS-Client API supports the ability to connect COBOL, PL/1, and Natural applications to DVM's virtualized data. The DS-Client requires modifications to the application and connection configuration for the AVZCLIEN subroutine.

A parameter list must be added to your application and used with subroutine calls to gain access and read/write data in this manner:

- ▶ OPEN
- ▶ SEND
- ▶ RECV
- ▶ CLOSE

OPEN creates a session with the DVM server and obtains shared memory that is used to send the SQL result sets to the application program. The SEND subroutine call is issued following an OPEN subroutine call to send the SQL request to the server. A RECV subroutine call is issued to retrieve the SQL results from the shared memory object created during the OPEN subroutine call. After the data is retrieved, a CLOSE subroutine call can be issued, which ends the session with the DVM server and releases resources.

2.10.4 z/OS Connect Enterprise Edition

z/OS Connect EE empowers a wide community of developers with a simple and intuitive way to use data and services on IBM Z through RESTful APIs. The DVM implementation can combine with IBM z/OS Connect EE for an easy-to-use interface in developing REST interfaces that access mainframe data sources, non-mainframe sources, such as SQL/NoSQL stores, and big data technologies, such as Spark and Hadoop. REST interfaces can be developed to combine multiple data sources. Developers do not see the underlying complexities of the data structures and their locations.

The z/OS Connect server has a DVM IBM WebSphere® Optimized Local Adapter (WOLA) service provider, which is an API that allows DVM to communicate, as shown in Figure 2-13.

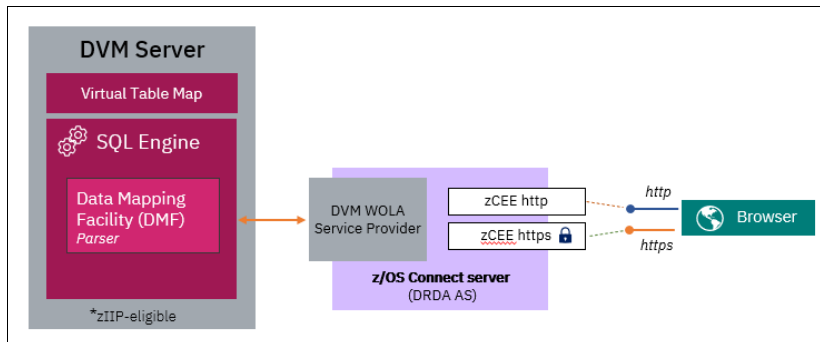


Figure 2-13 z/OS Connect Enterprise Edition and DVM WebSphere Optimized Local Adapter

Any virtualized data on the mainframe can be accessed through REST calls or through a web browser with the z/OS Connect Enterprise Edition server. The z/OS Connect server is deployed on the mainframe, separate from DVM, and listens to HTTP or HTTPS calls from the browser, or any HTTP clients.

2.10.5 Java Database Connectivity Gateway

DVM also includes the JDBC Gateway server (JGate), which is used to connect to distributed databases. It is installed separately on Linux, UNIX, Windows, or UNIX System Services systems. To access non-standard or non-DRDA-enabled data sources, the JGate server is needed, as shown in Figure 2-11 on page 24.

Specific relational database management systems (RDBMS) vendors require a specific licensing component that provides a DRDA-compliant data source. The DVM server connects to the JDBC Gateway server that uses a DRDA server. For example, Oracle connects by way of an Oracle JDBC driver. After the connection is established, data can be virtualized by using DVM Studio.

2.10.6 Connection and port security

When it comes to deployment, system administrators and system programmers can configure connections and port security. DVM supports Secure Sockets Layer (SSL) and is transparently supported by the Application Transparent Transport Layer Security (AT-TLS), an IBM TCP/IP facility. The DVM server recognizes and enables SSL connections and sessions automatically. Any port that includes a secure connection sends data in an encrypted format. DVM maintains the security and access credentials of the calling user that is based on the enterprise security manager in use (IBM RACF®, ACF2, and Top Secret).

SSL can be used to secure ODBC, JDBC, and HTTP network communications between the DVM server and DVM Studio. For HTTP, web services can be defined to support SSL by using the default setting of TLS 1.2.

For role-based authorization, users can access the DVM server by using AES, DVM domain-based, Microsoft Transaction Server, and Kerberos. The default authentication has a proprietary encryption mechanism when the login request is sent to the Host.

AES uses the Diffie-Hellman key exchange, and domain-based access verifies that the user is authenticated by a domain-based system. Windows platforms require that the user first logs on to an NT domain.

For the UNIX platforms, the local machine must be a member of a NIS domain. The password database that is used to authenticate the user must be NIS-mapped. Table 2-3 lists client endpoint characteristics and configuration information.

Table 2-3 DVM endpoint security configuration parameters

Endpoint	Connection method	Server configuration parameter	Secured login	Security PARM(INOO)
DVM JDBC/ODBC driver	Non-SSL	OEPORTRNUMBER (non-encrypted)	Yes (SAFUID/PWD)	
	SSL enabled	OESSLPORTNUMBER, SSL, SSLAUTODETECT, SSLCLIENTAUTH, SSLCLIENTNOCERT, SSLUSERID		
DVM Eclipse Studio	HTTP	WSOEPORTR (non-encrypted)	Yes (SAF UID/PWD)	
	HTTPS	WSOESSLPORT, SSL, SSLAUTODETECT, SSLCLIENTAUTH, SSLCLIENTNOCERT, SSLUSERID		
JGate server (DVM as client)	Non-SSL	DEFINE DATABASE PORT (<port number>) (non-encrypted)	Yes	DRDA, SECMEC ¹
	SSL enabled	DEFINE DATABASE PORT (<port number> SSL) (encrypted)		

¹ Security mechanism (SECMEC) is a technical tool that is used to implement a security service, such as access control lists, cryptography, and digital signatures.

Endpoint	Connection method	Server configuration parameter	Secured login	Security PARM(INOO)
DRDA (DVM as client)	Non-SSL	DEFINE DATABASE TYPE (MEMBER/GROUP/ZOSDRDA/ORACLE/MSSQL/QMFDRDA) PORT (<port number>) (non-encrypted)	Yes	DRDA, SECMEC ¹
DRDA (DVM as client)	SSL enabled	PORT (<port number> SSL) (encrypted)	Yes	DRDA, SECMEC ¹
Integrated DRDA Facility (ODF)	Non-SSL	IDF	Yes (SAF UID/PWD)	SECMEC ¹
	SSL enabled	IDFPORT (non-encrypted)		
zCEE WOLA DVM Service Provider	Non-SSL	HTTP Port inserver.xml	Yes (UID/PWD)	Liberty Server.xml httpPort
	SSL enabled	HTTPS Port in server.xml		
JDBC Gateway DRDA Listener	Non-SSL	Default configuration	Yes	DRDA, SECMEC ¹
	SSL enabled	JVM configuration for jsse2.overrideDefaultTLS JVM configuration for jsse2.overrideDefaultProtocol		
JDBC Gateway Administrator Console	Non-SSL	Port in jgate.properties	Yes (UID/PWD)	Creds stored in an application-managed file with encryption
	SSL enabled	HTTPS port in jgate.properties		

¹ Security mechanism (SECMEC) is a technical tool that is used to implement a security service, such as access control lists, cryptography, and digital signatures.

2.11 Summary

Because of its unique architecture, DVM supports real-time data virtualization technology that enables seamless access to mainframe relational and non-relational data for use with data analytics, big data, mobile, and web solutions.

DVM for z/OS is the only technology in the market that can be used by any client application program, loading process, or mainframe program with the possibility to read, update, or delete any data structure depending on the underlying data source.

All underlying data structures are abstracted from the user through the common usage patterns:

- ▶ Create virtual and integrated views of data to access mainframe data without having to move, replicate, or transform your source data.
- ▶ Enable access, update, and join functions on your mainframe data with other enterprise data with modern APIs in real time.
- ▶ Mask your mainframe data implementations from application developers to safely expose mainframe assets as APIs into mobile and cloud applications. Also, developers can use ANSI SQL functions to retrieve any data in mainframe and LUW environments.



Installation and configuration

IBM Data Virtualization Manager for z/OS (DVM for z/OS or DVM) is installed by using standard SMP/E methods that are common to other z/OS software product installations.

This chapter includes the following topics:

- ▶ 3.1, “Installation overview” on page 30
- ▶ 3.2, “Creating the DVM server data sets” on page 31
- ▶ 3.3, “Setting up the security application” on page 33
- ▶ 3.4, “Configuring the Workload Manager” on page 34
- ▶ 3.5, “Authorizing the program LOAD library” on page 34
- ▶ 3.6, “Creating a backup of the product libraries” on page 35
- ▶ 3.7, “Configuring support for the DBCS system” on page 36
- ▶ 3.8, “Customizing the DVM server for access to databases” on page 36
- ▶ 3.9, “Verifying the installation” on page 43

3.1 Installation overview

SMP/E is the basic tool for installing and maintaining software on z/OS systems and subsystems. It controls these changes at the element level by selecting the suitable levels of elements to be installed from many potential changes.

Complete the following steps to install DVM for z/OS:

1. Create the server data sets through the hlq.SAVZCNTL members AVZDFDIV, AVZGNMP1, and AVZEXSW1.
2. Set up the security application to use with the server by using one of the following hlq.SAVZCNTL members:
 - AVZRAVDB
 - AVZA2VDB
 - AVZTSVD
3. Configure Workload Manager (WLM) for optimum performance of the server.
4. Authorize the product LOAD library by using the Program Facility (APF).
5. Create a copy of the product libraries.
6. Configure the server to support DBCS to enable online use of databases.
7. Customize the server to access your data sources in hlq.SAVZEXEC(AVZSIN00).
8. Configure the started task JCL that is in hlq.SAVZCNTL(AVZ1PROC) before you can start the server.
9. Configure the Command List (CLIST) that starts ISPF panels by using hlq.SAVZEXEC(AVZ).
10. Verify the installation by creating a virtual table and accessing its underlying VSAM file.

3.2 Creating the DVM server data sets

The first step is to create DVM server data sets that are required for data processing. One data set is called a *global variable file* that retains parameters for the DVM server run-time execution. A server trace file can be used to record messages about server operations during processing.

Within the batch job that is called AVZDFDIV, you can change the product high-level qualifier and choose a four character server subsystem name that is unique for this server. The second and third character must be VZ, and the first and fourth characters can be anything that you decide. The example that is shown in Figure 3-1 is named AVZS.

```
VIEW          CSD.QA.AVZ0110.WRK.SAVZCNTL(AVZDFDIV) - 01.00  Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //JOBCARD == 0 ==> REPLACE THIS LINE WITH A VALID JOB CARD(S)
000200 //*****
000300 //*
000400 //* Copyright Rocket Software, Inc. 1991, 2017.
000500 //*
000600 //* Name - AVZDFDIV (was DEFDIV)
000700 //* Purpose - Define the Server Trace, Global Variable
000800 //* checkpoint and DMF VSAM data sets
000900 //* Related - Product started task JCL
001000 //* Parns - None
001100 //* Keywords - None
001200 //* Language - JCL
001300 //* Future - None
001400 //* History - 14 Dec 1992 PDS - Original implementation
001500 //* Notes - This member is used to create the Server
001600 //* Trace DIV data set and the global variable
001700 //* checkpoint data set.
001800 //*
001900 //* The following modifications should be made prior to
002000 //* executing this job:
002100 //*
002200 //* 0) Specify valid JOB CARD
002300 //* See "==" 0 ==>" above.
002400 //*
002500 //* 1) Change HLQ to the high level qualifier to use.
002600 //*
002700 //* 2) Change AVZS to the Server subsystem name.
002800 //*
002900 //* 3) If the site is not using SMS, add "VOL(volser)"
003000 //* to each DEFINE CLUSTER statement.
003100 //*
```

Figure 3-1 Job AVZDFDIV

Change HLQ1 to the HLQ of the product data sets (original SMP/e target), and change HLQ2 to the same value that you used in the AVZDFDIV member to denote the HLQ where the global variable and trace datasets are stored. This location, HLQ2 (can be server specific), is used to store the configuration datasets for each DVM SSID you define. For example, HLQ1 can be set to our SMP/e target lib example of DVM110, and HLQ2 can be set to DVM.AVZ1, so that the configuration that is specific to SSID AVZ1 is separated from the original SMP/e product lib DVM110

The second batch job to run is called AVZGNMP1 (see Figure 3-2) to generate a user-defined map file. As a user creates maps or edits existing maps, they are saved in the user-defined map file instead of being stored in the map file that is supplied by the product data sets.

```

VIEW          CSD.QA.AVZ0110.WRK.SAVZCNTL(AVZGNMP1) - 01.00  Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //JOBCARD == 0 ==> REPLACE THIS LINE WITH A VALID JOBCARD(S)
000200 //*****
000300 //*
000400 //*   * COPYRIGHT ROCKET SOFTWARE, INC. 1991, 2017.
000500 //*
000600 //* NAME           - AVZGNMP1 (WAS DEFMAP)
000700 //* PURPOSE        - DEFINE ADDITIONAL AVZ DATA SETS:
000800 //*                 - HLQ2.AVZS.SAVZMAP: USED TO STORE USER VIRTUAL
000900 //*                   MAPS.
001000 //*                 - HLQ2.AVZS.SAVZEXEC: USED TO STORE IN00 STARTUP
001100 //*                   REXX EXECS FOR ALL DATA SERVERS.
001200 //*                 - HLQ2.AVZS.SAVZX***: AVZS SPECIFIC EVENT FACILITY
001300 //*                   DATA SETS.
001400 //* RELATED      - PRODUCT STARTED TASK JCL
001500 //* PARS          - NONE
001600 //* KEYWORDS      - NONE
001700 //* LANGUAGE      - JCL
001800 //* FUTURE        - NONE
001900 //* HISTORY       - 14 DEC 1992 PDS - ORIGINAL IMPLEMENTATION
002000 //*                 28 AUG 2014 AAS - CHANGED MAP LRECL TO 2048
002100 //* NOTES        - THIS MEMBER IS USED TO CREATE ADDITIONAL AVZ
002200 //*                   DATA SETS.
002300 //*
002400 //*   THE FOLLOWING MODIFICATIONS SHOULD BE MADE PRIOR TO
002500 //*   EXECUTING THIS JOB:
002600 //*
002700 //*   0) SPECIFY VALID JOBCARD
002800 //*      SEE "== 0 ==>" ABOVE.
002900 //*
003000 //*   1) CHANGE HLQ1 TO THE HIGH LEVEL QUALIFIER OF THE SMPE
003100 //*      TARGET INSTALLATION LIBRARIES.
003200 //*
003300 //*   2) CHANGE HLQ2 TO THE HIGH LEVEL QUALIFIER OF THE AVZ
003400 //*      SUBSYSTEM DATA SETS.
003500 //*      -- CHECK HLQ2.AVZS.SAVZEXEC IN STEP IN00EXEC. THIS WILL BE
003600 //*           THE COMMON DATASET TO BE USED TO STORE ALL STARTUP

```

Figure 3-2 Job AVZGNMP1

Change HLQ1 to the high-level qualifier of the product data sets and change HLQ2 to a value to create the map file data set with.

Change AVZS to a wanted server subsystem name, (keeping in mind the xVZx rule). Anyone that uses this product requires UPDATE authority to this map data set so that maps can be created and then stored in the user-was defined map file.

Tip: Users need UPDATE authority to the MAP data sets that are created in this job.

3.3 Setting up the security application

The second step is to define security authorizations for the server that are based on the security application in use at your data center. You might need to involve your z/OS security specialists to ensure that you use the correct security application.

Run the following suitable job for your security application from within hlq.SAVZCNTL:

- ▶ AVZRAVDB: IBM Resource Access Control Facility (RACF) security
- ▶ AVZA2VDB: CA AFC2 security
- ▶ AVZTSVDB: CA top secret security

Figure 3-3 shows an example of the settings for RACF. Change WLMUSERID to the user ID you want to use to allow AVZ access to the WLM Policy definitions. This same user ID is used in the HLQ.SAVZEXEC.xxxxIN00 in the WLM parameter definitions. This permit can be removed if a user ID that includes the correct permissions is used.

```

BROWSE      ZOS.DATAVIRT.V110.SAVZCNTL.INSTALL (AVZRA Line 0000
Command ==>
//CRESTC   EXEC  PGM=IKJEFT01,DYNAMNBR=75,TIME=100,REGION=6M
//SYSTEM   DD  DUMMY
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//STDIN    DD  *
//STDENV   DD  *
//SYSTSIN  DD  *
PROFILE NOPREFIX
ADDUSER DVMADM NAME ('AVZ Server started task') -
  OMVS (HOME ('/u/dvmadm') PROGRAM ('/bin/sh') UID (1200)) -
  DFLTGRP (SYS1)

RDEFINE STARTED AVZ1.** LEVEL (0) OWNER (DVMADM) -
  STDATA (USER (DVMADM) GROUP (SYS1))
SETROPTS REFRESH RACLIST (STARTED)

PERMIT MVSADMIN,WLM.POLICY CLASS (FACILITY) ID (DVMADM) -
  ACCESS (READ)

```

Figure 3-3 RACF job example

In our example, we defined a new admin user ID of DVMADM, with its own OMVS segment. Change AVZS to your AVZ subsystem name (the default is AVZS for a new installation). We used AVZ1 to designate the DVM SSID we configured.

Also, the started task user ID requires READ, EXECUTE, READ, or READ/WRITE/UPDATE to the data sets that are listed in Table 3-1.

Table 3-1 Product data sets and their specific access privilege requirements

Data definition	Access	Data set name
STEPLIB	READ, EXECUTE	hlq.SAVZLOAD
AVZRPCLB	READ, EXECUTE	hlq.SAVZRPC
SYSEXEC	READ	hlq.SAVZEXEC
AVZTRACE	READ, WRITE	hlq.SAVZTRACE
AVZCHK1	READ, WRITE	hlq.SYSCK1
AVZMAPP	READ, WRITE	hlq.SAVZMAP

3.4 Configuring the Workload Manager

The third step involves configuring a suitable WLM policy for the DVM server. The following guidance can be used for the installation and customization. Most system programmers are familiar with the steps that are associated with WLM policy definitions.

A best practice is to configure the DVM server to use a medium to high performing WLM velocity goal, as its default service class. Complete the following steps:

1. Create a WLM classification rule:
 - a. From the WLM ISPF application, select option **6 (Classification Rules)**.
 - b. Select option **1** to create a rule.
 - c. Set the Subsystem Type to AVZ and provide an optional description.
 - d. When a default service class name does not exist, select option **4(Service Classes)** from the WLM menu and press **Enter** and **PF3** to save.
2. Define the DVM server started task AVZ1PROC to a WLM service class:
 - a. From the WLM ISPF application, select option **6 (Classification Rules)**.
 - b. For STC WLM-subsystem type, select **Modify**.
 - c. Add an entry for AVZ1PROC.
 - d. Provide a suitable service class for the started task and define it relative to workload management objectives.
 - e. Add a unique report class for the started task.
3. Activate the new WLM Policy definition.

3.5 Authorizing the program LOAD library

The next step is to APF-authorize the SAVZLOAD dataset from SDSF by using the **SETPROG APF** command.

Complete the following steps:

1. Verify the correct volume name from the Data Set List utility menu (see Figure 3-4).



```
DSLIST - Data Sets Matching ZOS.DATAVIRT          Row 35 of 59
Command --->                                     Scroll ---> CSR
Command - Enter "/" to select action             Message
Tracks %   XT Device  Dsorg Recfm Lrecl Blksz  Created  Referred
-----
ZOS.DATAVIRT.V110.SAVZLOAD                       U2640A
4125  60      1 3390   PD-E U                    0 32760 2020/10/20 2021/06/19
```

Figure 3-4 Verify the volume for SAVZLOAD

- From the SDSF main menu, enter command mode by entering / and then, pressing **Enter**. This mode gives you a bigger editor session to complete the next command. then, run the **SETPROG APF** command (see Figure 3-5).

```

System Command Extension
===> SETPROG APF,ADD,DSNAME=ZOS.DATAVIRT.V110.SAVZLOAD,volume=U2640A
===>
STORELIMIT
Comment

```

Figure 3-5 SETPROG APF command

- From SDSF, run the **APF** command from the main menu to see the list of APF authorized datasets (see Figure 3-6).

```

SDSF MENU V2R3M0 PLX62 STLAB62 LINE 1-19 (49)
COMMAND INPUT ==> APF SCROLL ==> CSR
NP NAME Description Group Status
DA Active users Jobs
I Input Queue Jobs
O Output Queue Output
H Held output Queue Output
ST Status of jobs Jobs

```

Figure 3-6 APF command

You should see SAVZLOAD listed, as shown in Figure 3-7.

```

COMMAND INPUT ==> - SCROLL ==> CSR
NP DSNAME Seq VolSer Status BlkSize Extent SMS LRecL DS
SYS1.DBLF.SDSNEXIT 77 STG001 MISSING 0 0 NO 0
SYS1.DBLF.SDSNLOAD 78 STG000 MISSING 0 0 NO 0
SYS1.DBLF.SDSNLOAD.V12 79 LAB620 MISSING 0 0 NO 0
SYS1.DBLF.SDSNEXIT.V12 80 LAB620 MISSING 0 0 NO 0
SYS1.DBLF.SDSNLOAD2.V12 81 LAB620 MISSING 0 0 NO 0
ZOS.DATAVIRT.V110.SAVZLOAD 82 U2640A OK 32760 1 YES 0 PO
ZOS.DATAVIRT.V110.SAVZRPC 83 U264F2 OK 32760 1 YES 0 PO

```

Figure 3-7 List of APF authorized datasets

3.6 Creating a backup of the product libraries

Next, a copy of the product libraries that are generated from the SMP/E installation must be created as a best practice. Also, a backup for recovery must be created when any customization must be retained. When you do not create a copy of your product data sets, any SMP/E maintenance you apply going forward overlays any customizations that are made.

Create the backup by using one of the following methods:

- ▶ Copy the product libraries to a runtime set of libraries.
- ▶ Run the IEFBR14/IEBCOPY job.
- ▶ Run the ADRDSSU DUMP job.

3.7 Configuring support for the DBCS system

In the next step, configure the initialization member, also known as the *server configuration member*, in the exec data set. The name of the member depends upon the four characters subsystem ID that you assigned the server. Therefore, we called it AVZS in our example; therefore, the initialization member is called AVZSIN00.

Consider the following points:

- ▶ HLQ.SAVZEXEC(AVZAIN00) is used to configure support for Japanese code pages and double-byte sets (see Figure 3-8).

```
/*-----*/
/* Set CCSID for non-DB2 data */
/*-----*/
If DoThis then
do
  "MODIFY PARM NAME (SQLENGDFLTCCSID) VALUE(1047)"
  "MODIFY PARM NAME (SQLENGDBCSTFMT) VALUE(ASIS)"
```

Figure 3-8 Configure support for DBCS

- ▶ SQLENGDFLTCCSID is the default CCSID to use when processing data by way of the server for non-Db2 data
- ▶ SQLENGDBCSTFMT is the default DBCS format to use when translating a single-byte code page to double-byte code page. Letters in single-byte code page are translated to full width DBCS characters.

3.8 Customizing the DVM server for access to databases

Customize the HLQ values (see Figure 3-4) in the initialization member to point to those libraries with which you created the product. HLQ1 is for the product target libraries that you created during the SMP/E installation process. HLQ2 represents the runtime libraries for the product. When runtime libraries are not created (as described in 3.6, “Creating a backup of the product libraries” on page 35), map HLQ2 to the product data sets that were created during the SMP/E installation process.

```
/*-----*/
/* Change SHLQ to your DVM installation high level qualifier */
/*-----*/
SHLQ = "CSD.QA.AVZ0110.WRK"
SHLQ2 = "CSD.AI38.AVZ9"
```

Figure 3-9 Customize the HLQ values

Modify the configuration member HLQ.SAVZEXEC(AVZSIN00) by providing the high-level qualifier of:

- ▶ Installation libraries in SHLQ
- ▶ Runtime libraries in SHLQ2

When no runtime libraries exist, SHLQ2 and SHLQ are recommended to have the same value.

During this process, you can customize the server initialization member for the TCP/IP ports that the server needs to communicate. You must have a dedicated web service port number (WSOEPOR) that the Eclipse plug-in can use to communicate with the DVM server (see Figure 3-10).

```

/*-----*/
/* Set the port numbers for OE sockets TCP/IP. These values are */
/* only used if you are running OE sockets. OE sockets can run */
/* over OE TCP/IP, z/OS TCP/IP, and other TCP/IP implementations as */
/* well. Both OEPORTNUMBER and WSOEPORT need to be set. */
/*-----*/
"MODIFY PARM NAME(OEPORTNUMBER) VALUE(1200)"
"MODIFY PARM NAME(WSOEPORT) VALUE(1201)"
"MODIFY PARM NAME(TRACEORW) VALUE(YES)"
"MODIFY PARM NAME(OEKEEPALIVETIME) VALUE(30)"
"MODIFY PARM NAME(PARALLELIO) VALUE(YES)"
"MODIFY PARM NAME(OEPIOPORTNUMBER) VALUE(1204)"

```

Figure 3-10 Customize the server initialization member for TCP/IP ports

3.8.1 Customizing the relational database

In this section, we present examples of how to set up the initialization member to access relational database management systems (RDBMS). Always add a DEFINE DATABASE section to the initialization file for RDBMS configurations. However, some data sources do not need any customization, such as VSAM. Data sources, such as Db2 distributed databases, Oracle, Teradata, or Amazon Redshift, require customization.

Figure 3-11, Figure 3-12, and Figure 3-13 show the initialization member configuration for DB2 for z/OS, DB2 LUW, and Oracle. A pattern exists for each defined database block with subtle changes that require database administrators for each data source to get specific values for the unique parameters for each.

```

J48200 "DEFINE DATABASE TYPE(MEMBER)"
J48300 "NAME(DB9A)"
J48400 "LOCATION(RS14DB9A)"
J48500 "DDFSTATUS(ENABLE)"
J48600 "DOMAIN(rs14.rocketsoftware.com)"
J48700 "PORT(5547)"
J48800 "CCSID(37)"
J49000 "IDLETIME(110)",
J49001 "APPLNAME(DB9ADB2)"

```

Figure 3-11 Initialization member for DB2 for z/OS

```

J47310 "DEFINE DATABASE TYPE(LUW)"
J47320 "NAME(DVS3)"
J47330 "LOCATION(DVS3)"
J47340 "DDFSTATUS(ENABLE)"
J47350 "DOMAIN(WAL-VM-SHDWDVS3)"
J47360 "PORT(50000)"
J47370 "CCSID(37)"

```

Figure 3-12 Initialization member for DB2 LUW

```

J01200 "DEFINE DATABASE TYPE(ORACLE)"
J01300 "NAME(ORDG)"
J01400 "LOCATION(DRDAAS1)"
J01500 "DDFSTATUS(ENABLE)"
J01600 "DOMAIN(WAL-VM-SHDWDVS2)"
J01700 "PORT(1446)"
J01800 "CCSID(500)"
J01900 "IDLETIME(0)"
J02000

```

Figure 3-13 Initialization member for Oracle

3.8.2 IMS database customization

IMS is customized differently from relational database management systems (see Figure 3-14).

```
100300 /*-----*/
100400 /* Enable IMS CCTL/DBCTL support */
100500 /*-----*/
100600 if 1 = 1 then
100700 do
100800 "MODIFY PARM NAME (DBCTL) VALUE (YES)"
100900 "MODIFY PARM NAME (IMSID) VALUE (IMS1)"
101000 "MODIFY PARM NAME (IMSMINTHREADS) VALUE (5)"
101100 "MODIFY PARM NAME (IMSMAXTHREADS) VALUE (10)"
101200 "MODIFY PARM NAME (IMSDSNAME) VALUE (IMS1.SDFSRESL)"
101300 "MODIFY PARM NAME (TRACEIMSDLIEVENTS) VALUE (YES)"
101400
101500 "MODIFY PARM NAME (IMSNBABUFFERS) VALUE (100)"
101600 "MODIFY PARM NAME (IMSFBBUFFERS) VALUE (10)"
101700 "MODIFY PARM NAME (IMSFPOVERFLOW) VALUE (10)"
101800 "MODIFY PARM NAME (TRACEIMSDLIEVENTS) VALUE (NO)"
101810 "MODIFY PARM NAME (SQLENGENABLESEGMAP) VALUE (YES)"
101811 /*-----*/
101812 /* ENABLE IMS-DIRECT SUPPORT */
101813 /*-----*/
101814 if 1 = 1 then
101815 do
101816 "MODIFY PARM NAME (ACIINTSEGMP256) VALUE (256)"
101817 "MODIFY PARM NAME (IMSDIRECTENABLED) VALUE (YES)"
101818 "MODIFY PARM NAME (IMSDIRECTMRDEFAULT) VALUE (8)"
101819 "MODIFY PARM NAME (IMSDIRECTBUFFER SIZE) VALUE (1024)"
101820 "MODIFY PARM NAME (TRACEIMSDBBREFRESH) VALUE (YES)"
101821 "MODIFY PARM NAME (TRACEIMSDBBLOCKS) VALUE (YES)"
101822 "MODIFY PARM NAME (TRACEIMSDIRSTATS) VALUE (YES) /* STATS */
101823 "MODIFY PARM NAME (TRACEIMSDIRSETUPBLKS) VALUE (YES) /* SETUP D */
101824 "MODIFY PARM NAME (TRACEIMSDIRTASKSETUP) VALUE (YES)"
```

Figure 3-14 Customization for IMS

Provide the IMS ID and the IMS SDFSRES library to access your IMS subsystem as a data source.

IMS access is broken down into two sections:

- ▶ The first section is used for CCTL or DBCTL support, which means you are making DLI transaction calls to IMS.
- ▶ The second section is for IMS Direct, which means we are accessing underlying files directly without making DLI calls. The IMS DBA can provide values to customize this block.

It is recommended that you collaborate with your IBM DBA to ensure data mappings and configuration settings are compatible.

3.8.3 Adabas customization

Customizing Adabas as a data source involves enabling an initialization member by setting the MODIFY PARM NAME (ADABAS) to YES. It is recommended that you collaborate with the ADABAS DBA to ensure that the data mappings and configuration are accounted for (see Figure 3-15).

```
080500 /*-----*/
080600 /* Enable Adabas database server support */
080700 /*-----*/
080800 if DoThis then
080900 do
081000 "MODIFY PARM NAME(ADABAS) VALUE(YES)"
081100 /*-----*/
081200 /* SET THE ADABASUINFOSIZE PARAMETER. PLEASE MAKE SURE THIS
081300 /* PARAMETER IS SET EQUAL OR GREATER THAN THE MAXIMUM OF THE
081400 /* USER INFORMATION SIZE AND REVIEW INFO SIZE IN YOUR ADALNK
081500 /* MODULE; OTHERWISE UNPREDICTABLE RESULTS CAN OCCUR.
081600 /* FOR MOST INSTALLATIONS, A VALUE OF 512 WILL BE SUFFICIENT.
081700 /*-----*/
081800 "MODIFY PARM NAME(ADABASUBINFOSIZE) VALUE(512)"
081900
082000 /*-----*/
082100 /* SET THE ADABASAUTOCOMMITBIND PARAMETER TO ENABLE AUTOCOMMIT
082200 /* FOR ADABAS CALLS. RECOMMENDATION - SET TO YES. BY SETTING
082300 /* THIS PARM TO YES, YOU WILL BE ISSUING AN ET (END TASK/TRANS.)
082400 /* IMMEDIATELY AFTER EACH UPDATE/STORE/DELETE TO THE DATABASE.
082500 /* PLEASE NOTE - THE PARAMETER DEFAULTS TO NO TO BE COMPATIBLE
082600 /* WITH OLDER APPLICATIONS DEVELOPED ON PREVIOUS RELEASES.
082700 /*-----*/
082800 "MODIFY PARM NAME(ADABASAUTOCOMMITBIND) VALUE(YES)"
082900 "MODIFY PARM NAME(ADABASDATEFORMAT) VALUE(OD)"
083000 end
083100
```

Figure 3-15 Customize for ADABAS

3.8.4 Configuring the started task JCL

After the DVM started task JCL is customized, it must be moved to the suitable procedure library. The started task is run immediately as the result of a **START** command and offers control over where and when the JCL is run.

In DVM for z/OS, the high-level qualifier of the target product installation libraries or the runtime libraries must be defined in the HLQ field.

Consider the following points:

- ▶ Db2 for z/OS requires the SDSNEZXIT and SDSNLOAD libraries to be added to the STEPLIB.
- ▶ IMS requires the IMS SDFSRESL library to be added to the STEPLIB concatenation.
- ▶ Adabas requires the ADABASE load library to be added to the STEPLIB.
- ▶ Ensure that the SYSEXEC DD statement allocates the correct data set name from the AVZSIN00 member.

The started task configuration is shown in Figure 3-16.

```

//AVZJ PROC SSID=AVZJ,
//          OPT=INIT,
//          TRACE=B,
//          MSGPFX=AVZ,
//          REG=0000M,
//          MEM=0200M
//          HLQ='CSJENN.RSQA.AVZ110'
*****
//IEFPROC  EXEC PGM=AVZ2IN,REGION=&REG,TIME=1440,DYNAMNBR=100,
//          MEMLIMIT=&MEM,
//          PARM='&OPT,&SSID,&TRACE,&MSGPFX'
//STEPLIB DD DISP=SHR,
//          DSN=RDSB.SDSNEXIT
//          DD DISP=SHR,
//          DSN=DSN.VA10.SDSNLOAD
//          DD DISP=SHR,
//          DSN=&HLQ..TESTPKG.SAVZLOAD
//          DD DISP=SHR,
//          DSN=IMS.IDA4.SDFSRESL
//          DD DISP=SHR,
//          DSN=CSD.SAG.SADALOAD
*****
//AVZRPCLB DD DISP=SHR,
//          DSN=&HLQ..TESTPKG.SAVZRPC
*****
//SYSEXEC DD DISP=SHR,
//          DSN=&HLQ..TESTPKG.SAVZEXEC
//SYSTSIN DD DUMMY,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//SYSTSPRT DD UNIT=VIO,SPACE=(CYL,(1,1)),DSN=&AVZTEMP,
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=10000)
//SYSOUT  DD SYSOUT=*

```

Figure 3-16 Started task configurations

Ensure that the SYSEXEC DD statement allocates the data set name that contains the initialization member for your customized data sets (see Figure 3-16).

```

//SYSOUT DD SYSOUT=*
*****
//* THE TRACE AND GLOBAL VARIABLES DIV DATA SETS CANNOT *
//* BE SHARED BETWEEN COPIES OF THE PRODUCT *
*****
//AVZTRACE DD DISP=OLD,
//          DSN=CSJENN.TCZ.TEST.&SSID..TRACE
//*
//AVZCHK1 DD DISP=OLD,
//          DSN=CSJENN.TCZ.TEST.&SSID..SYSCHK1
//*****
//* THIS DDNAME IS REQUIRED FOR DATA MAPPING FACILITY *
//*****
//AVZMAPP DD DISP=SHR,
//          DSN=CSJENN.AVZ.USER.MAP
//          DD DISP=SHR,
//          DSN=&HLQ..TESTPKG.SAVZMAP
//*****
//* THE CCTLSNAP DD IS REQUIRED FOR DUMPS FROM IMS CCTL CONNECTIONS *
//* CHANGE DUMMY TO YOUR SYSOUT DESTINATION TO RECEIVE THE DUMPS *
//*****
//CCTLSNAP DD DUMMY
***** Bottom of Data *****

```

Figure 3-17 Update SYSEXEC to point to the data set with the initialization member

Ensure that the AVZMAP concatenation contains the user-defined map data set as the first defined map dataset.

After these changes are made to the SYSEXEC file, start the DVM server for this example by using the /s AVZJ command.

3.8.5 Configuring the Command List

Configure the ISPF application (see Figure 3-18) to use the member AVZ in the SAVZEXEC data set of the fully qualified load library from the runtime library that you created in 3.6, “Creating a backup of the product libraries” on page 35. You can also use the product target library that was created by using the SMP/E installation process.

```

EDIT          RSTEST.DVM1.SAVZEXEC(AVZ) - 01.03          Columns 00001 00072
Command ==> |-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----|
003800 /*-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----|
003900 arg parm
004000 address TSO
004100
004200 LLIB = "RSTEST.DVM.SAVZLOAD"
004300
004400 address ISPEXEC
004500 "LIBDEF ISPLLIB DATASET ID('llib')"
004600 "SELECT CMD(AVZ "parm")"
004700
004800 address ISPEXEC
004900 "LIBDEF ISPLLIB"
005000
005100 exit
***** Bottom of Data *****

```

Figure 3-18 Configure the CLIST using ISPF

Complete the following steps to start the ISPF panels by using commands:

1. Edit hlq.SQVZEXEC member AVZ. Then, edit the parameter for your product load library llib='hlq.SAVZLOAD'.
2. Start the ISPF application by enter the following command on the ISPF command shell (see Figure 3-19):

```
EX 'hlq.SAVZEXEC(AVZ)' 'SUB(AVZS)'
```

```

Menu List Mode Functions Utilities Help
ISPF Command Shell
Enter TSO or Workstation commands below:
==> ex 'RSTEST.DVM1.SAVZEXEC(AVZ)' 'sub(avz1)'

Place cursor on choice and press enter to Retrieve command
=> IND$FILE PUT 'RSDEMO.ABP220.IBMTAPE.SABPSAMP.D092811(ABPBESTP)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB6)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB5)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB4)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB3)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB2)' ASCII CRLF
=> IND$FILE PUT 'CSJENN.ABP.CNTL(UETLAB1)' ASCII CRLF
=> IND$FILE GET 'DSN.ADB.SADBTLIB(ADB21A)' ASCII CRLF
=> IND$FILE GET 'DSN.ADB.SADBTLIB(ADB21T)' ASCII CRLF
=> ex 'ROCKET.USER.CLIST(HAAV41)'

```

Figure 3-19 Start the ISPF application

3.9 Verifying the installation

Use the Eclipse-based DVM Studio Client to verify the installation by creating a virtual table to access your underlying data files as a relational database format that uses SQL.

The detailed checklist can be used to assist you in gathering the information that you need to successfully install and configure the DVM server (see Table 3-2).

Table 3-2 Pre-customization tasks: DVM server (AVZ)

Asset type	Your value	Description
Started Task User ID		Grant security authority to write to its own data set (SAVZCNTL).
Administrator User ID		Grant security authority to write the SAVZCNTL and SACZMAP data sets.
Started Task User ID		User ID assigned to aVZ1PROC requires an OMVS segment defined.
Started Task name		Started Task User ID needs to be assigned or created and assigned.
User ID		User ID requires proper authority to run BINDS from the Studio.
IMS		IMS load library, PSB Library, and subsystem names.
Distributed RDBMS		User ID and password to read data, and the location, port, and IP.
Subsystem name		Assign the DVM server an unused subsystem and ID by way of TCz.
Started Task User ID		For RACF, a class must be defined to a descriptor table and system IPL'd. Start the AVZ1PROC server. RACF access granted to specific data sets.
Db2 for z/OS		RACF PassTickets must be created per the user guide
Library		Eclipse plug-in Users must update their authority to the user map data sets specified in the AVZ1 started task.
Port		Obtain the TCP/IP OE port number (Open Edition [OE]).
Port		Obtain a Web service port number that the Eclipse plug-in can use to communicate.
Ports		If TCP/IP ports are protected, the AVZ1 subsystem name needs to have proper access NOT the AVZ1PROC Started Task.
User		During installation of DVM Studio Eclipse plug-in, the user must be signed on to DVM Studio as Administrator.



Connecting to z/OS data sources

In this chapter, we discuss how the DVM server can access data on the mainframe by using mainframe database systems, file management systems, or system files.

This chapter includes the following topics:

- ▶ 4.1, “Introduction” on page 46
- ▶ 4.2, “Getting started” on page 47
- ▶ 4.3, “Direct access to z/OS databases” on page 47
- ▶ 4.4, “Db2 for z/OS” on page 52
- ▶ 4.5, “IBM ESA/IMS database” on page 54
- ▶ 4.6, “Accessing mainframe files” on page 56

4.1 Introduction

The DVM server has built-in intelligence to use direct-access paths to underlying data that is on the mainframe environment, such as mainframe database systems, file management systems, or system files.

The benefits are faster access and run time, and the ability to more readily offload general processing to zIIP specialty engines, if available. Clients access the DVM server through ODBC/JDBC, HTTP/HTTPS, DRDA, including RRSAF and CAF for Db2 z/OS (see Figure 4-1).

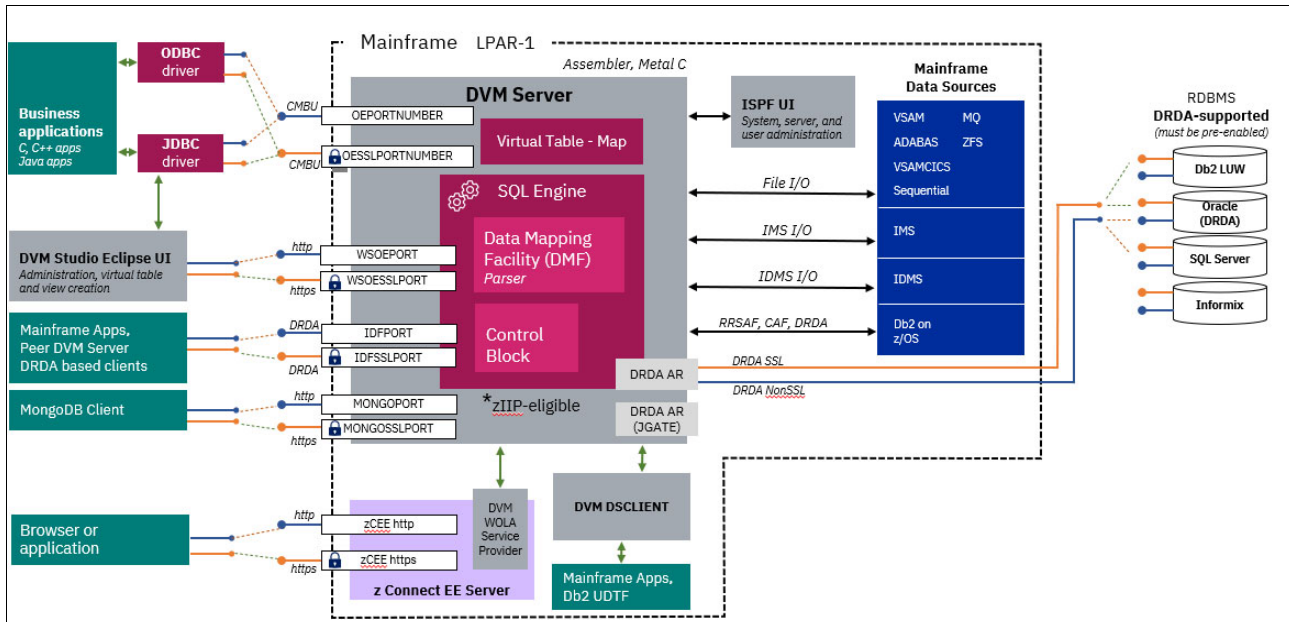


Figure 4-1 Client access methods to the DVM server

4.2 Getting started

Within the DVM configuration member, specify the DRDA RDBMS settings through a definition statement and provide local environment values for all the parameters. The DVM Started Task must be recycled after the Define Database specifications are set.

Example 4-1 shows the section in the configuration member to enable.

Example 4-1 Define database specifications

```
"DEFINE DATABASE TYPE(type_selection)" ,  
"NAME(name)" ,  
"LOCATION(location)" ,  
"DDFSTATUS(ENABLE)" ,  
"DOMAIN(your.domain.name)" ,  
"PORT(port)" ,  
"IPADDR(1.1.1.1)" ,  
"CCSID(37)" ,  
"APPLNAME(DSN1LU)" ,  
"IDLETIME(110)"
```

For more information about installing and configuring the DVM server, see [IBM Documentation](#).

4.3 Direct access to z/OS databases

DVM for z/OS provides direct access to popular z/OS databases, such as Db2 for z/OS, IMSDB, and Adabas. The DVM server evaluates and processes optimized query access, which in some cases (depending on the data source) can avoid management subsystems for I/O.

4.3.1 ADABAS

Adabas is designed to support thousands of users in parallel with subsecond response times. Access to Adabas can be from Natural, Software AG's 4RL dev IDE, ODBC/JDBC, and embedded SQL. Adabas supports up to 65,535 DBs with each supporting up to 5,000 files, where each file can contain up to 4 trillion records, each with up to 926 fields. DVM for z/OS supports Adabas 8.x or later and the DVM server supports long names for fields, Multi-file joins optimized connection modes, Natural subsystem security, and dynamic switching between TCB and SRB mode for improved parallelism and zIIP-eligibility (Figure 4-2). The DVM server supports ET/BT transaction-based commands, MU and PE file structures, and Natural DATE and TIME formats.

Benefits of using DVM for z/OS to access Adabas

The use of DVM for z/OS to access Adabas realizes the following benefits:

- ▶ Software AG's Adabas SQL requires an off-host server (CONNX) with limits for scale, performance, and failover that can be addressed with DVM server technology.
- ▶ The DVM server can service both Adabas and Natural programs and serve as a substitute for Software AG's EntireX solution.
- ▶ DVM for z/OS is an integral component for IBM's IDAA Loader component for loading Adabas data to IBM's Data Analytics Accelerator (IDAA).

- ▶ DVM for z/OS DScient supports DRDA and the JDBC Gateway server. This support offers a replacement option to Software AG’s Natural SQL Gateway.
- ▶ Organizations with Db2 for z/OS can drive transaction activity through their Db2 subsystem by using the Integrated DRDA Facility (IDF) that uses Db2 as a Data Hub. This option isolates workloads to a single system.
- ▶ DVM for z/OS integrates with AT-TLS and improves configuring secure access to Adabas data.
- ▶ DVM for z/OS helps to offload SORT, JOIN, and data translation operations.

Adabas can run a single Adabas command that retrieves multiple rows (multi-fetch). When a SELECT statement runs within the DVM server, the request can be split into multiple and separate READ requests when MapReduce is dynamically activated and returns a list of record IDs (ISN), as shown in Figure 4-2.

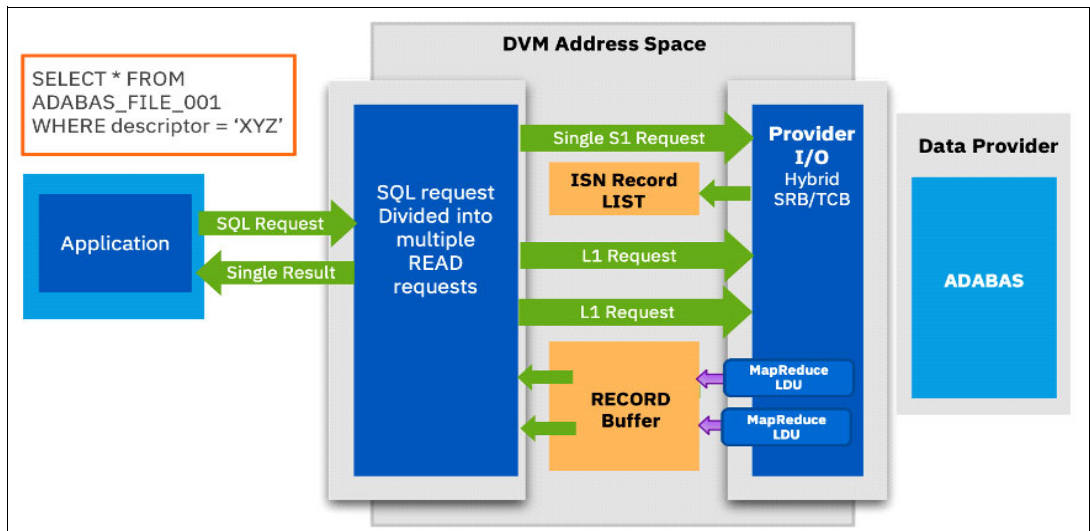


Figure 4-2 DVM server architecture for Adabas

Virtualizing Adabas

Typically, any older data source for virtualizing DVM uses the physical file and a COBOL or PL/1 copybook to map the meta-data into DVM format (DVM Catalog tables). However, in the case of Adabas, this procedure is slightly different. Adabas uses a data definition module (DDM) to create a view of the Adabas file, manage long column names, or limit the view to a subset of defined fields. The DDM represents the metadata for the Adabas data, which is analogous to catalog tables for a relational database.

The virtualization procedure in DVM for z/OS for Adabas is a manual procedure and is run by using JCL (which is provided with the DVM server) or can be used by using DVM Studio.

Creating virtual tables with JCL

DVM for z/OS is packaged with Batch JCL, where the administrator selects one file number at a time. Batch JCL parameters include; Subsystem, SSID, DBID, and File Number. Running the Batch JCL generates metadata in XML and then parsed by the DVM parser to create DVM server metadata (see Figure 4-3).

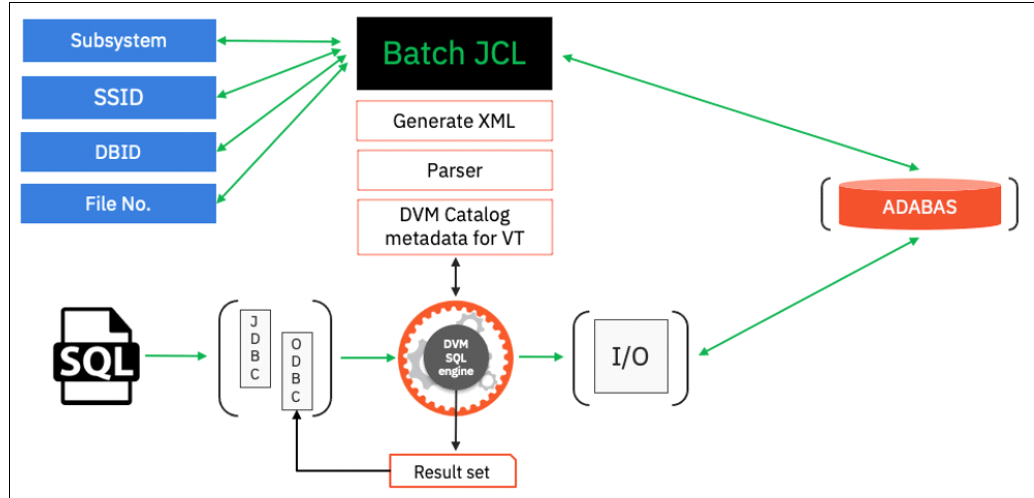


Figure 4-3 Virtualizing Adabas with Batch JCL

Working with Adabas

Adabas features basic fields, special fields, and multi-value fields. These fields must be translated by the DVM server and transformed collectively into a relational format for client tools to more easily access and query the underlying data. The DVM catalog stores the metadata for virtual tables. Table 4-1 lists basic fields format conversions.

Table 4-1 Basic field format conversions for Adabas

Field format	Length	SQL data type
Binary	126 bytes	Binary
Fixed	2 bytes	Small integer
Fixed	4 bytes	Integer
Fixed	8 bytes	Big integer
Float	4 bytes	Real float
Float	8 bytes	Double
Packed	29 bytes	Decimal
Unpacked	29 bytes	Decimal or numeric
Alpha	253 bytes	Varchar

Table 4-2 lists special field format conversions.

Table 4-2 Special field format conversions for Adabas

Field format	Length	SQL data type
ISN	RECORD_ID	Big integer
Natural/Predict (D) date	4 bytes	Date
Natural/Predict (T) time	7 bytes	Timestamp

The DVM server also works with Multi-value (MU) fields in an Adabas record and periodic groups (PE), which are repeating groups in an Adabas record. Both multi-value field and periodic groups are limited to 191 occurrences and a maximum fetch of 64 K occurrences. If Adabas applications run a delete, the Adabas data set collapses the array of data by removing the deleted record and reassigning the next record in its place.

When a virtual table is flattened, each multi-value occurrence becomes a column. When the table is not flattened, each multi-value field is written its own subtable with an index, parent key, and base key (see Figure 4-4).

	ADDRESS_LINE	ROW_INDEX	PARENT_KEY	BASE_KEY
530	AN DER BAHN 34	1	00000036	00000036
531	4590 CLOPPENBURG	2	00000036	00000036
532	GERHARDTSTR. 78	1	00000037	00000037
533	6100 DARMSTADT	2	00000037	00000037

Figure 4-4 Multi-value fields in a flattened virtual table

When working with periodic groups, one level of multi-value (MU) fields can be inside of a PE, to make a 3-D array. Deleting an occurrence does *not* collapse the array. When the virtual table is flattened, each field in the PE is a column. When the virtual table is not flattened, the PE is a subtable with a group of columns with an index and parent key.

Best practices for field naming

Adhere to the following best practices when naming fields:

- ▶ Adabas by design includes field names that are two characters and it is recommended to use longer field names.
- ▶ Generate Natural DDM views as an Adabas DBA.
- ▶ When flattening ME or PU arrays, limit the field name to 26 bytes to account for extra bytes that are used during data mapping.

Best practices for load testing

Each DVM ODBC/JDBC client connection performs an (OP)en for an Adabas User Queue Element (UQE). These UQEs share a pool of Adabas Nucleus Threads (NT) that run on the General Processors.

The DVM server takes advantage of Adabas v8 multi-buffers and Adabas 64-bit storage through multiple fetches of records for better performance and reduced CPU. Depending on the type of transaction, the size of the format buffer and result-set (record buffer), Adabas manages the workload based on the ADARUN parameter settings that are listed in Table 4-3.

Table 4-3 ADARUN parameters

Parameter	Definition	Recommendation
V64BIT	YES	A setting of YES activates 64-bit storage.
NU	Maximum # of user queue elements.	Consider increasing this parameter for a larger number of application connections (for example, 800 for a total of 800 user connection pool).
LU	All user buffers (format, record, search, value, and ISN). Recommended value of 1024000.	LU represents all buffers that might be required for any Adabas command: <ul style="list-style-type: none"> ▶ Increase this value to improve performance and ability to handle large PE groups. ▶ Minimum value of 64 K. ▶ Recommend 1024 K.
NAB	The number of attached buffers to be used. Recommended value of 10000.	Increase this value with corresponding increases in LU.
LWP	The size of the Adabas work pool. Recommended value of 3,500,000.	Increase this value with corresponding increases in LU.
NISNHQ	The maximum number of records that can be placed in HOLD status at the same time by one user.	Typically one quarter of the NH HOLD queue size. Important for large updates; for example, up to and exceeding 12000.
NH	HOLD queue element (HQE)	This setting is required for each record (ISN) placed in HOLD status. This value is important for large updates; for example, up to and exceeding 50000.
LBP	The maximum size of the Adabas buffer pool.	Monitor Adabas shutdown stats to size this value for performance; for example, 1,200,000,000.
NTS	The number of user threads used during the Adabas session.	Typically, corresponds to the number of General Processors + 1.

Generating code for Adabas

DVM for z/OS provides natural code generation, which can be used to access Adabas. The DVM server supports Software AG Natural version 4 and later.

DVM Studio provides an option for compiling the generated natural program outside of the mainframe. After virtual tables are defined for an Adabas data set, by right-clicking the **Virtual Tables** section and selecting **Generate Query with ***, the administrator can quickly generate code that can be used by a client application to access the data set.

In DVM Studio, select **Server Tab** → **SQL** → **Data** → **SSID** → **Virtual Tables** → **Generate Query with *** (see Figure 4-5).

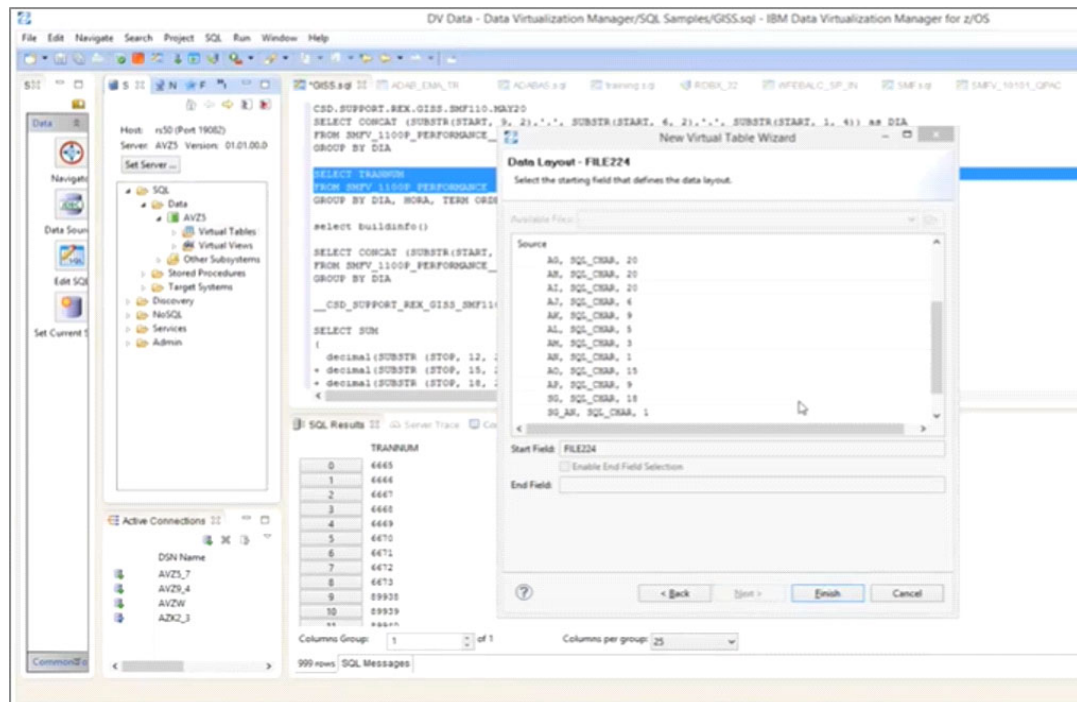


Figure 4-5 Adabas code generator

The IBM Documentation provides detailed

For more information about this configuration process, see [this web page](#).

4.4 Db2 for z/OS

A user-defined-table-function (UDTF) is a function that is defined within a Db2 systems by using the Create Function statement and can be accessed from SQL. DVM provides the ability to define UDTF's on virtual tables and virtual views, which allows Db2 applications the ability to access DVM's virtual data real-time.

UDTFs are programs that can reach outside of Db2 for z/OS to the operating system or file system to retrieve data and return it in a relational format. DVM for z/OS can create Db2 for z/OS UDTFs that point to virtual tables provisioned on the DVM server.

IDF enables DVM to be defined as an application server within Db2 communication database.

4.4.1 Db2 for z/OS access options

Db2 for z/OS can be accessed by the DVM server by using the Distributed Relational Database Architecture (DRDA) access method or the Resource Recovery Services attachment facility (RRSAF) access method.

The DRDA method allows a higher percentage of the Db2 workload to run in Service Request Block (SRB) mode and be offloaded to a zIIP specialty engine. Running workloads in SRB mode lowers the total cost of ownership when compared to RRSAF by reducing the dependency on z/OS General Purpose processor use (MIPS).

If the z/OS environment uses a zIIP specialty engine, configure the DVM server to access Db2 for z/OS with the DRDA access method. Before Db2 requests are issued, you must bind DRDA, RRSAF, or both into packages within each Db2 subsystem. Binding both access methods is recommended.

Two Db2 for z/OS data access constructs are available by the Data Virtualization Manager server:

- ▶ Traditional Db2 APIs allows for reading and writing the data, and transactional integrity.
- ▶ Db2 Direct reads the underlying Db2 VSAM linear data sets directly, without issuing an SQL statement against Db2 for z/OS. This access method allows read-only access to the data and provides high performance and bulk data access.

This method can be used when you create virtual tables against Db2 database objects with DVM Studio. In a controlled environment, the Db2 Direct access was used on a large z14 configuration to virtualize and facilitate large data pulls that resulted in greater than 5 times improvement in elapsed time when compared to traditional DRDA access. In this case, the work is 100% zIIP eligible compared to 60% eligible where Db2 uses DRDA requesters. This access method does not use any Db2 resources (see Figure 4-6).

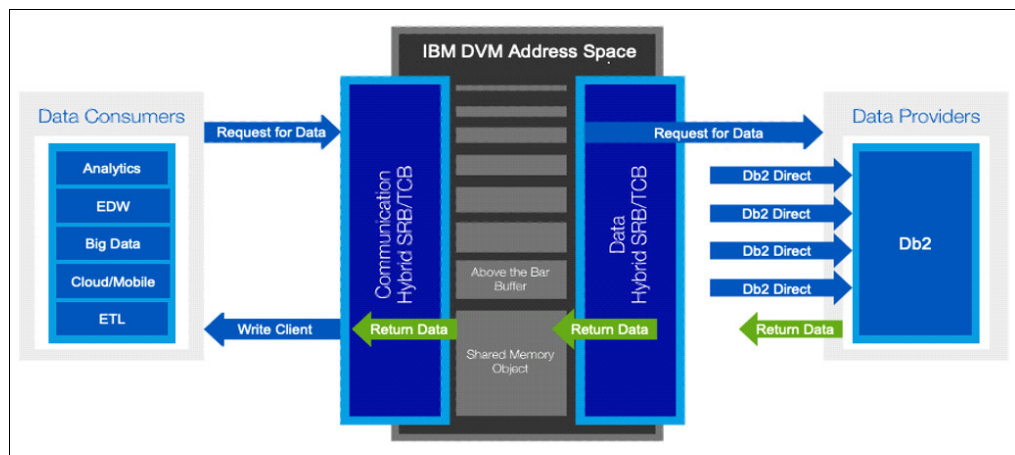


Figure 4-6 Db2 Direct provides direct access to underlying data without DB2 resources

For more information about configuring access to Db2 for z/OS, see [this web page](#).

4.5 IBM ESA/IMS database

IBM IMS database support is provided through three access methods that use simple SQL-based queries, as described next.

4.5.1 IMS database control

Database Control (DBCTL) is an IMS facility that provides an IMS Database Manager (IMS DM) subsystem that can be attached to CICS but runs in its own address spaces. DBCTL access supports SELECT INSERT UPDATE and DELETE.

Note: ODBA and DBCTL are mutually exclusive. Enable only one of these two methods to access IMS.

4.5.2 IMS Direct

The DVM server directly accesses underlying native data sets and bypasses the IMS management software for savings on general CPU usage. This access method eliminates the need to traverse the IMS Subsystem with no locking involved when accessing the data. Direct access in this fashion does not fully secure data integrity if the application is performing updates and deletes of data.

Security is managed on the IMS native data set when IMS Direct is used. The user ID of the client connection must have the necessary security permissions for reading the IMS database data sets, as shown in Figure 4-7.

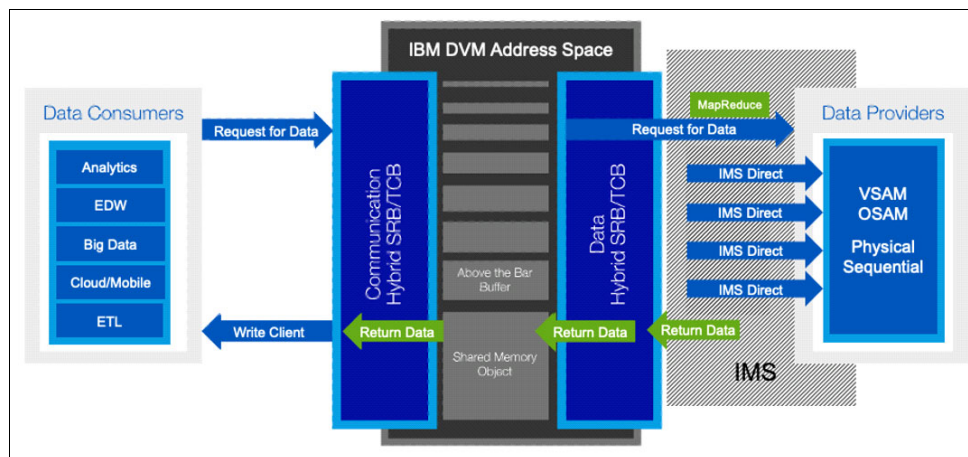


Figure 4-7 IMS Direct is fully zIIP-eligible and faster for direct access to IMS data

When IMS Direct is unavailable, the DVM server uses the DBCTL access method. Statistics about the IMS database are collected and stored within a metadata repository from which the SQL engine optimizes the MapReduce process and dynamically determines which is the most performant method for access based on the SQL query. IMS Direct supports all IMS database types, except SHISAM and HISAM:

- ▶ Hierarchical direct access method (HDAM): VSAM and OSAM
- ▶ Hierarchical indexed direct-access method (HIDAM): VSAM and OSAM
- ▶ Partitioned HDAM (PHDAM): VSAM and OSAM

- ▶ Partitioned HIDAM (PHIDAM): VSAM and OSAM
- ▶ Fast Path data entry database (DEDB)

IMS Direct supports SELECT Only for bulk data access and low general processor usage. Multiple IMS subsystems can be accessed by using this method.

4.5.3 IMS Open Database Access

IMS Open Database Access (ODBA) provides a callable interface that enables any z/OS recoverable, resource-managed z/OS address space, including the DVM server, to issue DL/I database calls to an IMS DB subsystem. The interface provides access to full-function DL/I databases and data entry databases (DEDBs).

ODBA supports SELECT, INSERT, UPDATE, and DELETE and two-phase commit. Multiple IMS subsystems can be accessed by using this method. When configuring multiple IMS subsystems, enable DBCTL and IMS Direct or ODBA and IMS Direct.

Note: ODBA and DBCTL are mutually exclusive. Use only one of these two methods to access IMS for a single DVM server.

4.5.4 Configuring IMS

As with other data sources, there are two data sets that are supplied by DVM for z/OS that must be edited to configure IMS. To configure access, the DVM server started task JCL and configuration member hlq.xVZy.SAVZEXEC.(xVZyIN00) must be modified. More configuration changes are necessary to the IMS system.

4.5.5 Creating virtual tables

The Program Specification Block (PSB) and Database Definition (DBD) source members and the copybooks for each IMS segment must exist in the virtual source libraries that are defined to the server. For more information, see [this web page](#).

4.5.6 Enabling IMS Direct

Enable the IMSDIRECTENABLED parameter configuration member. When an IMS SQL query is run, the SQL Engine determines whether the request is best run with IMS Direct (native file support) or if IMS APIs are required. This determination is based on the supported database and file types, and the size of the database.

The PSB and DBD source members and the copybooks for each IMS segment must exist in the virtual source libraries that are defined to the server. For more information, see [this web page](#).

In the IN00 configuration member, search for the following sections to enable access methods:

```
DoThis = 1
DontDoThis = 0
...

/* Enable IMS CCTL/DBCTL support */
if DontDoThis
then
do MODIFY PARM NAME(DBCTL) VALUE(YES) "
...
```

4.6 Accessing mainframe files

Much of critical business data on the mainframe exists in file formats and data sets that are not managed by any specific database management system. While these files lack the DBMS to ensure data integrity, the DVM server can access the files through its Data Mapping Facility (DMF) and transform them into a normalized relational view for SQL access.

Consider the following points:

- ▶ Sequential files feature records in the file in the order that they are written and can be read back in the same order only. They are a simple form of a COBOL file. Every record in a relative file can be accessed directly without reading through the other records. After it is virtualized, the DVM server allows this sequential data to be retrieved in any group order, along with query predicates.
- ▶ Delimited files typically are not found in a mainframe environment, but can be in any file storage system and include column delimiters for a stream of records that consist of fields that are ordered by column. Each record contains fields for one row, with individual fields that are separated by column delimiters. The DVM server requires the use of rules to virtualize delimited files.
- ▶ System Management Facility (SMF) offers a way of keeping track of what is occurring on your mainframe. A total of 255 SMF records are available to help deliver information about all functions and products that are running on the z/OS environment from the machine level to application-specific activity that is related to processing, I/O, allocated memory, and so on.
- ▶ Syslog is a standard for computer message logging and can be used for computer system management, security auditing, analysis, and debugging messages.
- ▶ Log stream is an application-specific collection of data that oftentimes organizations use to optimize their environment and routinely store in the form of SMF records. Log stream data sets are VSAM linear data sets.
- ▶ Operations log (OPERLOG) is a sysplex-wide log of system messages that are in a system logger log stream, which is similar to Syslog.

4.6.1 VSAM

VSAM is a type of data set and an IBM DASD file storage access method that is used to store mission-critical data and managed by a file management system. The problem is that modern applications want to access this data more readily, but VSAM is not a database.

DVM for z/OS provides seamless access to native VSAM data without configuring for SQL access, other than creating a virtual source library that maps DVM server metadata to the native data that is on disk that describes the record structure in copybooks. This mapping helps to correctly read the data. The DVM server supports COBOL, PL/I, and DCSCECT formatted copybooks.

This section divides this process into steps to help simplify this process into the following building blocks that are required to virtualize VSAM data with DVM Studio:

- ▶ A VSAM cluster that represents the VSAM data set, including one or both of data or index data records. The DVM server requires the name of a valid VSAM cluster as part of the source library for mapping purposes. No other configuration steps are required for the DVM server to set up the SQL interface to native VSAM files.
- ▶ The data set Member name that is included in the virtual source library that maps to the copybook (record layout) for the underlying data structure residing on disk.

The process involves the one-time creation of a virtual source library by containing metadata that is needed to correctly access and read VSAM data, followed by the creation of a virtual table, which allows for SQL access by any number of client applications.

Creating a virtual source library

DVM Studio is instrumental in creating the building blocks for the DVM server to use in virtualized data. All data sources require a virtual source library to be configured to virtualize underlying data. The library structure ensures a valid description and mapping for the DVM server for locating, accessing, and processing SQL statements.

DVM Studio allows a user to create a virtual source library under the Admin folder of the Server Tab for Source Libraries.

In DVM Studio, select **Server Tab** → **Admin** → **Source Libraries** → **Create Virtual Source Library** (see Figure 4-8).

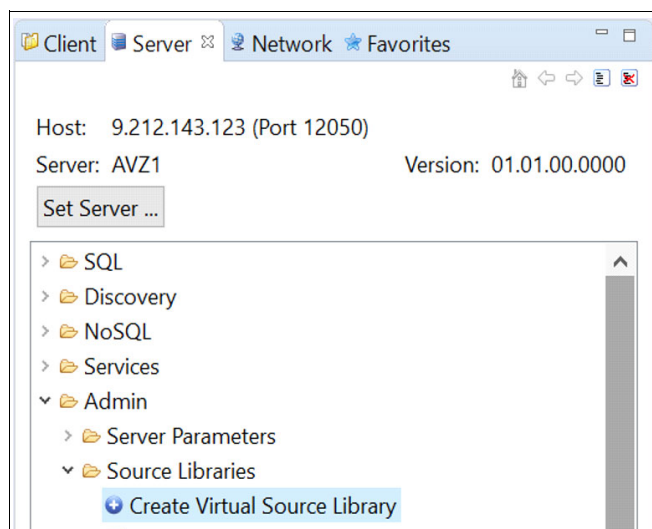


Figure 4-8 Creating a virtual source library

The Virtual Source Library wizard requires a name, description, and the selected host library name that includes the VSAM record layout from a drop-down list of available source libraries. The underlying data structure can be virtualized into a virtual table for clients to access and query.

Creating a virtual table

DVM Studio allows a user to create a virtual table under the SQL folder of the Server Tab for Virtual Tables. This step requires the name of the DVM subsystem (SSID) that includes the metadata and virtual source library for the wanted data set.

In DVM Studio, select **Server Tab** → **SQL** → **Data** → **SSID** → **Virtual Tables**.

Right-clicking the Virtual Tables label opens a Virtual Tables wizard to help define criteria for access, fields to include in the virtual table, and allows for the ability to validate the virtual table definition. In this example, it ensures that the VSAM cluster name and VSAM cluster type match and display the suitable success or failure.

The Virtual Tables wizard prompts you to select the data type; then, it advances to the New VSAM Virtual Table dialog window for more definitions and requires that the data mapping library that is defined during the configuration of the DVM server as part of the started task JCL be selected.

The accessing USER and SERVER must include READ/WRITE permissions to the target data set, as shown in Figure 4-9.

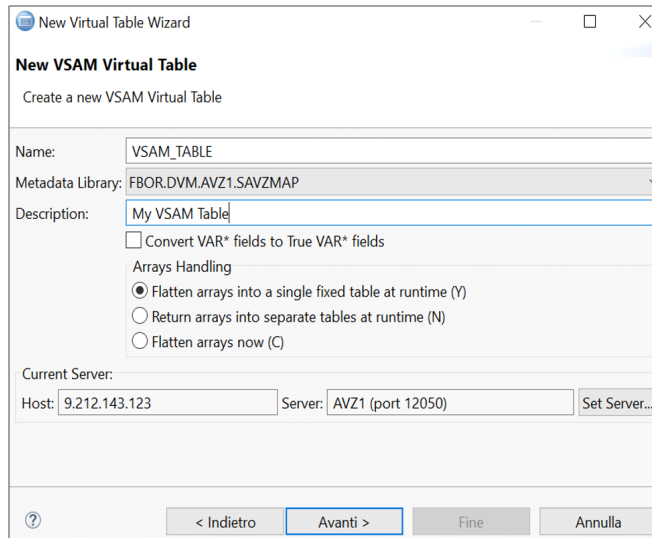


Figure 4-9 Creating a virtual table with DVM Studio Wizard

The Virtual Tables wizard then requests the administrator to select the previously defined virtual source library that contains copybooks for available data. After the suitable source library is downloaded and selected, a list of copybook members appears for selection.

Selecting the suitable copybook member that is associated with the VSAM data set allows the administrator to advance through the wizard and define the table layout when querying the data, shown in Figure 4-10 on page 59.

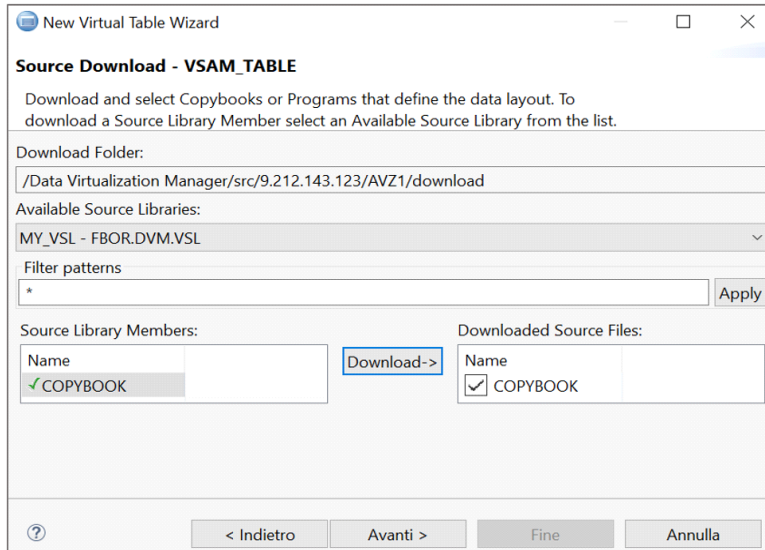


Figure 4-10 Associating the source copybook to virtual table

If the last field from the source file's Copybook is required, the **Enable End Field Selection** option must be selected, as shown in Figure 4-11.

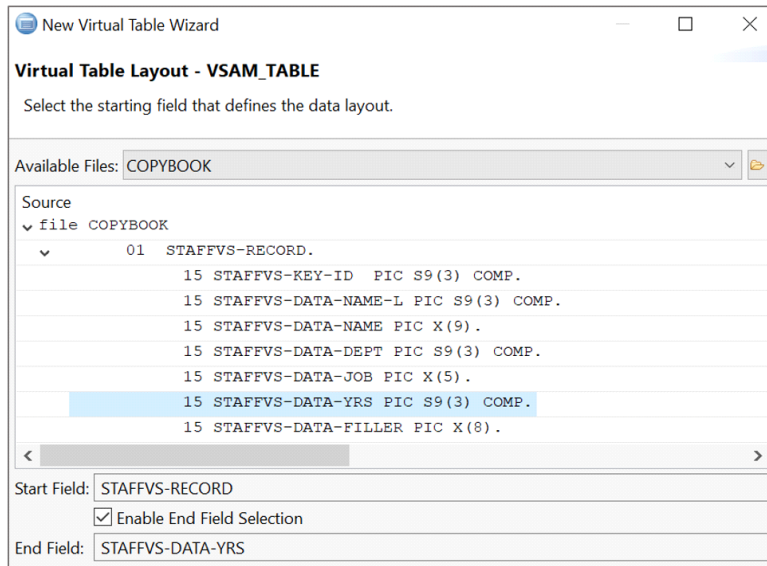


Figure 4-11 Virtual table layout

The virtual table definition must be validated before creation. In this example, the VSAM cluster name matches the type for the underlying file (KSDS), as shown in Figure 4-12.

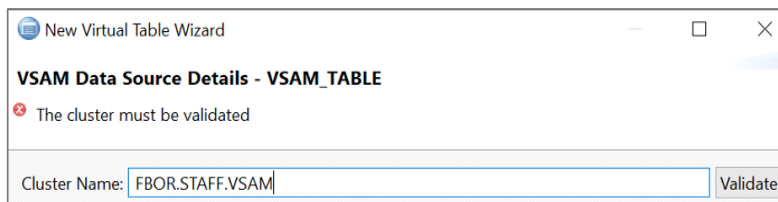


Figure 4-12 Associating VSAM cluster name to virtual table

The DVM server checks the VSAM cluster name that is provided and returns a confirmation message with the type of VSAM cluster (see Figure 4-13).

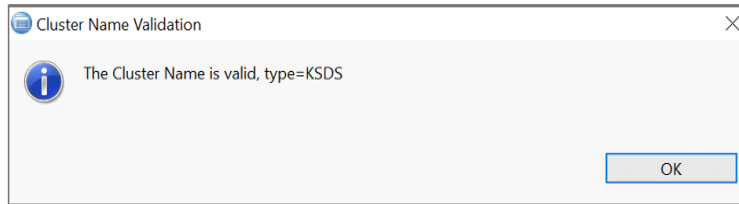


Figure 4-13 Successful validation of VSAM cluster

Querying the new virtual table

With the new virtual table created, right-clicking the virtual tables label again and selecting **Generate Query with *** allows the administrator to quickly verify that data can be retrieved without error. With this step complete, the administrator is ready for the next step in provisioning the new object for test, development, or production-ready applications to use, as shown in Figure 4-14.

In DVM Studio, select **Server Tab** → **SQL** → **Data** → **SSID** → **Virtual Tables** → **Generate Query with ***. This selection auto-generates a scripted query statement with details about data set type, retrieved rows, and location of the data. The generated SQL statement can be modified inline or copied and use as part of application development. The following SQL output provides a sample view of how the auto-generated SQL looks:

```
-- Description: Retrieve the result set for OPERLOG_SYSLOG (up to 1000 rows)
-- Tree Location: 10.3.58.61/1200/SQL/Data/AVZS/Virtual Tables/OPERLOG_SYSLOG
-- Remarks: Logstream - SYSPLEX.OPERLOG
SELECT * FROM OPERLOG_SYSLOG WHERE SYSLOG_JOBID = 'AVZS';

-- Description: Retrieve the result set for VSAM_TABLE (up to 1000 rows)
-- Tree Location: 10.3.58.61/1200/SQL/Data/AVZS/Virtual Tables/VSAM_TABLE
-- Remarks: VSAM - FBOR.STAFF.VSAM
SELECT * FROM VSAM)_TABLE LIMIT 1000;
```

	STAFFVS_KEY_ID	STAFFVS_DATA_NAME_L	STAFFVS_DATA_NAME	STAFFVS_DATA_DEPT	STAFFVS_DATA_JOB	STAFFVS_DATA
0	10	7	SANDERS	20	MGR	7
1	20	6	PERNAL	20	SALES	8
2	30	8	MARENGHI	38	MGR	5
3	40	7	O'BRIEN	38	SALES	6
4	50	5	HANES	15	MGR	10
5	60	7	QUIGLEY	38	SALES	0

Figure 4-14 Result of SQL query on VSAM virtual table VSAM_TABLE

4.6.2 System and operations logging

As workload processes on the mainframe, z/OS communicates status and problems through messages that it writes to logs. Busy systems can generate a large amount of data, which is written to SYSLOG and OPERLOGS. Many organizations began forwarding this data to Splunk for analysis.

In this case, DVM for z/OS can be used to virtualize selective logs for local analysis or forward those logs to Splunk, where the DVM server can virtualize that data and blend it with other local data. Many users choose to use IBM Common Data Provider for z Systems® (CDPz).

Although other options are available, some are costly from a licensing perspective and require more capital outlay from an infrastructure standpoint regarding CPU, storage, and memory, staging environments, and so on. It is for this reason that DVM for z/OS becomes an option with no data movement and in-place access to the system and operational data.

The DVM server supports over 35 data sources, including SMF, SYSLOG, OPERLOG, and JOBLOG and can serve in a multi-purpose manner for various use cases.

OPERLOG is a merged, sysplex-wide system message log that is provided by a log stream of data. SYSLOG contains a partition (LPAR) message log and is an SYSOUT data set that is produced by JES2 or JES3. DVM for z/OS provides five predefined virtual tables to display OPERLOG and SYSLOG:

- ▶ OPERLOG_SYSLOG accesses the SYSPLEX log stream and is defined in the global variable GLOBAL2.SYSLOG.DEFAULT after the AVZSYSLG rule is enabled
- ▶ OPERLOG_MDB
- ▶ OPERLOG_MDB_MDB_CONTROL_OBJECT
- ▶ OPERLOG_MDB_MDB_TEXT_OBJECT
- ▶ SYSLOG

Configuring access for SYSLOG

To configure access to system log (SYSLOG) files, the use of the DVM server configuration member AVZSIN00 and built-in VTB rules are required. VTB rules are provided to define the SYSLOG data set name. Each of the rules for SYSLOG processing requires that table names for use by SQL begin with SYSLOG.

The following rules are provided:

- ▶ AVZSYSLG uses a global variable to specify the name of the data set to use for the SYSLOG data.
- ▶ AVZSYSL2 supports the use of generation data group (GDG) data set names. One of the following formats is expected and the general use of global variables allows for maximum flexibility in the overall configuration:
 - SYSLOG_GDG_####, where #### is a relative GDG number (0 - 9999) that is appended to the GDG base name value that is obtained from the GLOBAL2.SYSLOG.GDGBASE variable.

For example, if the table name as specified in the SQL statement is SYSLOG_GDG_1, the data set name that is returned by this rule is HLQ.SYSLOG(-1), depending on the value in GLOBAL2.SYSLOG.GDGBASE.
 - SYSLOG_DSN_suffix, where suffix is used as the last part of a global variable of the form GLOBAL2.SYSLOG.suffix to look up the name of the data set to be used. If this variable does not exist, the data set name is specified in GLOBAL2.SYSLOG.DEFAULT is used to read the SYSLOG records.

Global variable examples possible for use with this rule:

- GDGBASE hlq.SYSLOG
- DEFAULT hlq.SYSLOG(0)
- TODAY hlq.SYSLOG(0)
- YESTERDAY hlq.SYSLOG(-1)

Customizing rules for SYSLOG

After the VTB rules for SYSLOG are enabled, they persist for every occurrence of an SQL statement where the use of SYSLOG as a prefix for table names in SQL statements. To enable VTB events for SYSLOG, the DVM server configuration member must be customized by configuring the SEFVTBEVENTS parameter in the AVZSIN00 member from the DVM server ISPF PANEL.

```
"MODIFY PARM NAME(SEFVTBEVENTS) VALUE(YES)"
```

When configuring AVZSYSL2 for SYSLOG, no customization of the VTR is required. However, when configuring for AVZSYSLG, specify S next to the AVZSYSLG in the ISPF PANEL and change the data set name to SYSLOG.

Enabling rules for SYSLOG

Enable AVZSYSLG and AVZSYSL2 rules by specifying E next to the member in the ISPF PANEL. To auto-enable these rules after each DVM server restart, specify A next to the member name instead. If the event global variables are needed, the administrator must configure the SYSLOG global variable, shown in Figure 4-15.

```
----- Event Facility (SEF) Event Procedure Lis Row 19 to 22 of 22
LCs: S ISPF Edit E Enable D Disable A Set Auto-Enable
      Z Reset Auto-Enable B Set Auto/Enable C Disable/Reset Auto

PDS Members for: DVM.ZB01.AVZS.SAVZXVTB
          A
S Member Status E TYP VV.MM Created Modified Size Init Mod ID
-----
a AVZSYSLG ENABLED Y VTB 01.00 20/04/30 20/04/30 15:03 81 81 0 ARNOULD
a AVZSYSL2 ENABLED Y VTB 01.00 20/05/19 20/05/19 17:46 146 146 0 ARNOULD
```

Figure 4-15 Auto-Enabling AVZSYSLG and AVZSYSL2 VTB rules

DVM Studio can now be used similar to running sample queries on VSAM by selecting SYSLOG in the virtual tables section of the Server Tab. Figure 4-16 shows the generated SQL statement and the results.

SYSLOG_SEQUENCE_NUMBER	SYSLOG_RECORD_TYPE	SYSLOG_REQUEST_TYPE	SYSLOG_ROUTING_CODES	SYSLOG_SYSTEM_NAME	SYSLOG_DATE_TIME	SYSLOG...
0	N		C020000	ZB01	2020/05/14 1...	CICSRJ
1	N		C020000	ZB01	2020/05/14 1...	CICSRJ
2	N		C020000	ZB01	2020/05/14 1...	CICSRJ
3	N		C020000	ZB01	2020/05/14 1...	CICSRJ
4	N		C020000	ZB01	2020/05/14 1...	CICSRJ
5	N		C020000	ZB01	2020/05/14 1...	CICSRJ
6	N		FFFFFFF	ZB01	2020/05/14 1...	CICSRJ
7	N		C020000	ZB01	2020/05/14 1...	CICSRJ
8	N		C020000	ZB01	2020/05/14 1...	CICSRJ
9	N		C020000	ZB01	2020/05/14 1...	CICSRJ
10	N		C020000	ZB01	2020/05/14 1...	CICSRJ
11	N		C020000	ZB01	2020/05/14 1...	CICSRJ

Figure 4-16 Generated query and results for SYSLOG with DVM Studio

In DVM Studio select: **Server Tab** → **SQL** → **Data** → **SSID** → **Virtual Tables** → **SYSLOG** → **Generate Query with ***.

The following query is generated and the output resembles the output that is shown in Figure 4-16:

```
-- Description: Retrieve the result set for SYSLOG (up to 1000 rows)
-- Tree Location: 10.3.58.61/1200/SQL/Data/AVZS/Virtual Tables/SYSLOG
-- Remarks: HTTP://10.3.58.61:1201
SELECT * FROM SYSLOG LIMIT 1000;
```

Configuring access to OPERLOG

No modifications are needed to configure the DVM server to access OPERLOG data; however, OPERLOG must be active in a system logger log stream. Use the IBM mainframe's System Display and Search Facility (SDSF) to verify whether OPERLOG is active with the '/D C,HC' inputs. OPERLOG is actively configured and enabled if the following message is displayed:

```

CNZ4100I 15.19.16 CONSOLE DISPLAY 056
CONSOLES MATCHING COMMAND: D C,HC
MSG:CURR=0 LIM=9000 RPLY:CURR=0 LIM=9999 SYS=P02
PFK=00
HARDCOPY LOG=(SYSLOG,OPERLOG) CMDLEVEL=CMDS
ROUT=(ALL)
LOG BUFFERS IN USE: 0 LOG BUFFER LIMIT: 9999

```

DVM Studio can now be used similarly to run sample queries on VSAM by selecting **OPERLOG_SYSLOG** in the virtual tables section of the **Server** tab. Figure 4-17 shows the first 100 rows.

SYSLOG_SEQUENCE_NUMBER	SYSLOG_RECORD_TYPE	S...	SYSLOG_ROUTING_CODES	SYSLOG...	SYSLOG_DATE_TIME	SYSLOG_JOBID	SYSLOG_REQ
1734	N		8000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000090
1735	N		0000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000290
1736	N		8000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000090
1737	N		0000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000290
1738	N		0000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000290
1739	N		8000000	ZB01...	2020/05/05 19:09:52.0...	AVZS	00000090
1740	N		8000000	ZB01...	2020/05/05 19:09:53.0...	AVZS	00000090
1741	N		8000000	ZB01...	2020/05/05 19:10:32.1...	AVZS	00000090
1742	M		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1743	D		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1744	D		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1745	M		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1746	D		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1747	D		0040000	ZB01...	2020/05/05 19:13:27.1...	AVZS	00000290
1748	N		8020000	ZB01...	2020/05/06 15:13:44.1...	AVZS	00000090

Figure 4-17 Generating Query on OPERLOG with DVM Studio

In DVM Studio select: **Server tab** → **SQL** → **Data** → **SSID** → **Virtual Tables** → **OPERLOG_SYSLOG** → **Generate Query with ***.

We can verify access to OPERLOG data by issuing the following query, a limit value was added to the end of the query:

```
SELECT * FROM OPERLOG_SYSLOG LIMIT 1000;
```

These results are similar to the SYLOG example; however, records exist for a different ZB02 LPAR because we are now reporting across the sysplex. Querying on the ZB01 LPAR allows us to test and validate only in an interactive mode in the ISPF PANEL for the DVM server. Replace sentence with:

We can add predicates to our previous example to query for the DVM server with more specificity by using the AVZS JOBID:

```
SELECT * FROM OPERLOG_SYSLOG
WHERE SYSLOG_JOBID='AVZS';
```

The query results for the AVZS subset are identical to the results that are shown in Figure 4-17 because they are ordered by the SYSLOG_JOBID column.

In the DVM server ISPF PANEL in interactive mode, the Address Environment can be updated to AVZ to generate a message in the OPERLOG by running the **DISPLAY REMOTE USER(*)** command, as shown in Figure 4-18.

```

----- SEF - Command Response Display ----- Row 5 to 8 of 8
ADDRESS Environment ==> AVZ      (SEF, AVZ., TSO, or REXX)
Environment Command ==> DISPLAY REMOTE USER(*)
-----
* No output was queued *
* 1 AVZLINE.n variables returned *
ARNOLD DESKTOP-4NU9I6U OTC/IP No :;FFFF;10.32.56.170 61599 448 -1 2 008C5BE0
----- Return code 0 from AVZ cmd "DISPLAY REMOTE USER(*)" -----
**End**

```

Figure 4-18 Displaying remote users

When running the same query again in DVM Studio, the SYSLOG_DATE_TIME value is incremented, as AVZS generates a new message in OPERLOG (see Figure 4-19):

```

-- Description: Retrieve the result set for OPERLOG_SYSLOG (up to 1000 rows)
-- Tree Location: 10.3.58.61/1200/SQL/Data/AVZS/Virtual Tables/OPERLOG_SYSLOG
-- Remarks: Logstream - SYSPLEX.OPERLOG
SELECT * FROM OPERLOG_SYSLOG WHERE SYSLOG_JOBID = 'AVZS';

```

JUTING_CODES	SYSLOG_SYSTEM_NAME	SYSLOG_DATE_TIME	SYSLOG_JOBID	SYSLOG_REQUEST_FLAGS	SYSLOG_MSG_TEXT	LS_TIMESTAMP
1736	ZB01	2020/05/05 1...	AVZS	00000090	AVZ4391I OE stack bi...	2020-05-...
1737	ZB01	2020/05/05 1...	AVZS	00000290	AVZ4273H OE sockets ...	2020-05-...
1738	ZB01	2020/05/05 1...	AVZS	00000290	AVZ4273H OE sockets ...	2020-05-...
1739	ZB01	2020/05/05 1...	AVZS	00000090	AVZ4391I OE stack bi...	2020-05-...
1740	ZB01	2020/05/05 1...	AVZS	00000090	AVZ3255I SQL DB2 key...	2020-05-...
1741	ZB01	2020/05/05 1...	AVZS	00000090	AVZ0362I No active s...	2020-05-...
1742	ZB01	2020/05/05 1...	AVZS	00000290	IXG231I IXGCONN REQU...	2020-05-...
1743	ZB01	2020/05/05 1...	AVZS	00000290	SUCCESS FOR JOB AVZS...	2020-05-...
1744	ZB01	2020/05/05 1...	AVZS	00000290	DIAG1: 00000000 DIA...	2020-05-...
1745	ZB01	2020/05/05 1...	AVZS	00000290	IXG232I IXGCONN REQU...	2020-05-...
1746	ZB01	2020/05/05 1...	AVZS	00000290	SUCCESS FOR JOB AVZS...	2020-05-...
1747	ZB01	2020/05/05 1...	AVZS	00000290	DIAG1: 00000000 DIA...	2020-05-...
1748	ZB01	2020/05/06 1...	AVZS	00000090	*AVZ3127S VSAM CHECK(...	2020-05-...
1749	ZB01	2020/05/06 1...	AVZS	00000090	AVZ3259E (DISPLAY) i...	2020-05-...
1750	ZB01	2020/05/06 1...	AVZS	00000090	AVZ3259E ("DISPLAY) ...	2020-05-...

Figure 4-19 SQL Query result from OPERLOG with recent messages

4.6.3 Delimited file data sets

The most common form of delimited data is CSV file or Microsoft Excel worksheet. When delimited data processing is activated through VTB rules, processing occurs in columnar order. The delimited data must include a value for each column in the map in the correct order to prevent errors.

To enable delimited data processing, the DVM server configuration member (AVZSIN00) must be customized by configuring the SEFVTBEVENTS parameter:

```
"MODIFY PARM NAME(SEFVTBEVENTS) VALUE(YES)"
```

The DVM server ISPF panel allows you to customize a sample rule that is named AVZMDDL from the VT rule management section. Within this section, column and string delimiter values, and control header processing can be enabled.

The vtb.optbdlcv option must be set to 1:

```

/*-----*/
/*   Activate delimited data processing for the table           */
/*-----*/
vtb.optbdlcv = 1          /* flag that data is delimited.   */

```

The following options are available to assist with processing delimited data:

- ▶ vtb.optbdlco sets the column delimiter (default value is the comma “,”).
- ▶ vtb.optbdlch sets the delimiter (default is the double quotation mark “”).
- ▶ vtb.optbdlhr identifies and removes the header record containing column names. If specified without a header prefix, the system compares the first token in each line to the first column name in the table to recognize and discard the header. The default is no header checking with value 0.
- ▶ vtb.optbdlhp is a global parameter that defines prefix data that identifies the beginning of a header line to be discarded. The specified value can contain a maximum of 32 bytes. This value is compared to the beginning of each delimited line of data before any tokenization is performed.

Defining map definitions for delimited data sets

To read a delimited data set, data type mappings must be in place for the delimited file we want to virtualize. Figure 4-20 shows a sample CSV file that is in the local zFS file system.

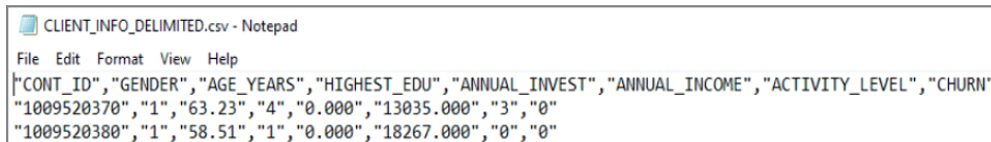


Figure 4-20 DCL definitions for CLIENT_INFO delimited data set

Map definitions are needed to ensure that columns are displayed in the correct order. The process is similar to that performed for VSAM or other input files. Figure 4-21 shows a copybook definition of the delimited file data set that declares the field definitions to be created in the DVM server virtual source library.

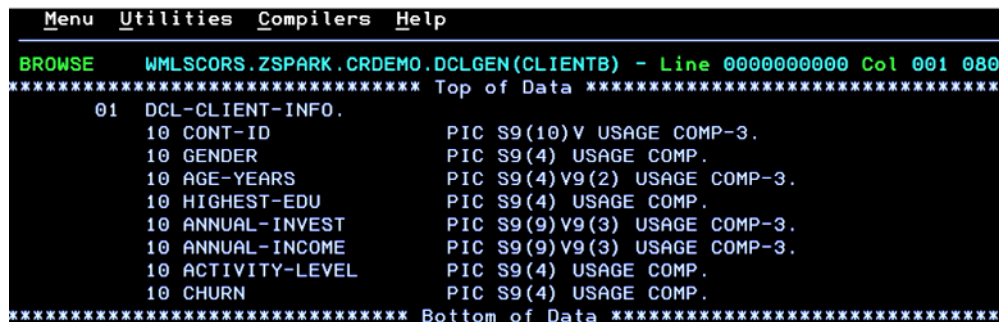


Figure 4-21 DCL definitions for CLIENT_INFO delimited data set

DVM Studio can now be used in a similar way to running sample queries on VSAM by selecting zFS in the Virtual Tables section of the Server tab. Figure 4-22 on page 66 shows the generated SQL statement and results.

	CONT_ID	GENDER	AGE_YEARS	HIGHEST_EDU	ANNUAL_INVEST	ANNUAL_INCO...	ACTIVITY_LEVEL	CHURN
0	1009520370	1	63.23	4	0.000	13035.000	3	0
1	1009520380	1	58.51	1	0.000	18267.000	0	0
2	1009520390	0	31.10	2	111192.000	129276.000	2	0
3	1009520400	0	49.84	1	0.000	17867.000	2	0
4	1009520410	1	53.33	1	0.000	17576.000	1	0
5	1009520420	1	47.72	3	90419.000	111569.000	5	0
6	1009520430	0	48.46	2	11258.000	20142.000	1	1
7	1009520440	0	34.38	1	0.000	33727.000	2	0
8	1009520450	1	47.09	1	0.000	17678.000	2	0

Figure 4-22 Result from SQL Query on Delimited data set

Be careful to add the MDDL_M_ prefix before the Virtual Table name to format data correctly for display. The MDDL_M_ prefix is required for SQL to ensure correct formatting.

In DVM Studio, select: **Server tab** → **SQL** → **Data** → **SSID** → **Virtual Tables** → **zFS** → **Generate Query with *** to result in the following query:

```
-- Description: Retrieve the result set for CLIENT_INFO_DELIMITED (up to 10000
rows)
-- Tree Location: 10.3.58.61/1200/SQL/Data/AVZS/Virtual
Tables/CLIENT_INFO_DELIMITED
-- Remarks: zFS file - /u/arnould/CLINET_INFO/DELIMITED.csv
SELECT * FROM MDDL_M_CLIENT_INFO_DELIMITED LIMIT 10000;
```

Data conversion errors occur if the delimited data is not compatible with the host types of the columns. If the conversion fails, diagnostic information that is related to the error is automatically logged for troubleshooting problems.

4.6.4 System Management Facility

IBM z/OS System information can be logged by using the IBM System Management Facility (SMF) and the native DVM server logging feature. Logging allows you to collect various systems and operations-related information.

The following IBM APARs must be applied on the z/OS SMP/E base system:

- ▶ APAR OA49263 provides real-time SMF support and is a requirement for the configuration of real-time SMF data access.
- ▶ APAR OA48933 is required to address accessing log streams. SMF log stream configuration is required for in-memory resource support. In this section, we cover how to access this information from the DVM server.

The following methods are available to access to SMF files:

- ▶ SMF data sets: SMF information is recorded in MANx data sets. When a data set is full, the data is processed by using IFASMFDP. The output of IFASMFDP is required when global variables are used.
- ▶ Log streams: SMF information can be recorded in multiple log streams and determined by the data set name beginning with IFASMF, which is used by the VTB rule for SMF.
- ▶ In-memory SMF data offers real-time access and can be read directly from the z/OS system buffer.

Upon DVM server initialization, SMF connects to the in-memory resource and continuously reads a buffer of SMF activity by using a REXX procedure. The REXX procedure is responsible for reading data set names from in-memory objects.

From a DVM server perspective, the SMF data set is driven by the Server Event Facility (SEF) rules. SEF rules are provided with default values in member h1q.AVZS.SAVZXVTB (AVZSMFT1). It is used when a table with the prefix SMF_TYPE_ is found in the SQL statement.

It also is used to specify the base map name and the data set name for SMF tables in a global variable. A data set name can be specified for a specific table (SMF record type) by creating a global variable for the table name. This allows applications to use other SMF data sources without exposing their names.

This REXX procedure provides the name of data sets or in-memory objects that must be read (a global variable that is named VTB.OPTBDSNA is going to be completed at execution).

Note: You must at least define the global variable GLOBAL2.SMFTBL2.DEFAULT to make these rules work. These VTB rules also can be customized according to your needs and naming conventions.

To configure access to SMF files, you must configure the server started task JCL, server configuration member, and server virtual table member. To enable reading SMF data in real time in log streams, you must have the SMFPRMxx member in the system PARMLIB data set that is configured to use both log streams and in-memory resources.

SMF data set names are dynamic in local environments and require SEF rules enablement and optionally global variables set to specific values to provide data set names to the virtual tables and views from SMF data sets or log stream configurations.

You can choose a GDG data set name to support or dynamic data set name support, or both, to quickly access your SMF data. These two options are provided for your convenience to help you start accessing your SMF data. Custom rules likely must be developed to use your local naming convention to access your SMF files. It is common to use GDG data sets to automatically export SMF data to disk from a fixed GDG base name.

SMF from GDG data sets

Enable read access of SMF data from GDG data sets, and access to SMF data through dynamic data set names, by enabling Data Virtualization Manager Server Event Facility rule AVZSMFT1.

On the DVM server, select **Rules Mgmt** → **SEF Rule Management** → **VTB** → **Enable** → **Auto-enable**.

To configure the access method, complete the following steps:

1. In the global variables display of the DVM server ISPF panel, update the global prefix to GLOBAL2.
2. Configure the SMF data access for the SMFTBL2 data set. The DEFAULT variable should have a corresponding SMF dump data set name if used. This option is used to specify the source SMF:

```
GLOBAL2.SMFTBL2.DEFAULT = "YOUR.DATASET.SMF.GDG"
```

This syntax is useful if we want to read the FULL GDG data set.

Pro tip: Be careful to define the data set name when uppercase is used. If you specify the correct name, but in lower or mixed cases, the allocation fails.

To filter out or select specific GDG members, other variables must be configured. For example, to add TODAY, in the command line of the global prefix GLOBAL2.SMFTBL2, enter S TODAY and exit edit mode, as shown in the following example:

```
GLOBAL2.SMFTBL2.TODAY = "YOUR.DATASET.SMF.GDG(+0)" (for today's GDG only)
GLOBAL2.SMFTBL2.YESTERDAY = "YOUR.DATASET.SMF.GDG(-1)" (for yesterday's GDG only)
```

Next, we test that the definitions are correctly defined by running a query against our GDG data set. In Data Studio, which was used earlier to access VSAM files, we edit and run the following query:

```
SELECT * FROM SMF_07001_YESTERDAY
```

Then, we see result that is shown in Figure 4-23 when we run the query.

	SMF_LEN	SMF_ZERO	SMF_FLAG	SMF_RTY	SMF_TIME	SMF_SID	SMF_SSI	SMF_STY	SMF_SEQN	SMF70RTY	SMF70TME	SMF70DTE	SMF70SID	SMF70S
0	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	30000	2020125	ZB01	RMF
1	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	60000	2020125	ZB01	RMF
2	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	90000	2020125	ZB01	RMF
3	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	120000	2020125	ZB01	RMF
4	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	150000	2020125	ZB01	RMF
5	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	180000	2020125	ZB01	RMF
6	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	210000	2020125	ZB01	RMF
7	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	240000	2020125	ZB01	RMF

Figure 4-23 Result from SQL Query on SMF70 records from Yesterday

If we want to change the default GDG data set name, we can change the VTB global variable or submit the new GDG data set name in the SQL Query. To pass a dynamic data set name to query an SMF data set, we use the following format for the table name in the SQL statement:

```
- TableMapName__DataSetName
```

Where:

- ▶ - TableMapName is SMF_07001
- ▶ - DataSetName is prefixed by two underscores (__) and the periods in the data set name are replaced with single underscores (_).

Edit and run the following SQL from Data Studio to get the results that are shown in Figure 4-24. To display SMF records from "Today", run the following Select statement and use parenthesis in the SQL and double quotation mark for the table name:

```
SELECT * FROM "SMF_07001__SMF_RECORDS_ZB01_SMF_SAVE(+0)"
```

	SMF_LEN	SMF_ZERO	SMF_FLAG	SMF_RTY	SMF_TIME	SMF_SID	SMF_SSI	SMF_STY	SMF_SEQN	SMF70RTY	SMF70TME	SMF70DTE	SMF70SID	SMF70S
0	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	30000	2020126	ZB01	RMF
1	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	60000	2020126	ZB01	RMF
2	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	90000	2020126	ZB01	RMF
3	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	120000	2020126	ZB01	RMF
4	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	150000	2020126	ZB01	RMF
5	23868	0	11011111	70	2020...	ZB01	RMF	1	...	70	180000	2020126	ZB01	RMF

Figure 4-24 Result from SQL Query on SMF70 records from a dynamic data set name

SMF from log stream

Another way to read SMF data is to connect to a log stream to make available more real-time SMF Data. Enable rule AVZSMFT1 and add the following global variable to the existing variables:

```
GLOBAL2.SMFTBL2.LOG = "LOGSTREAM.dataset.name"
```


Figure 4-25 shows the current SMF PARMLIB member that is associated with the DVM server. The SMFPRMxx member can be modified to make any change to the SMF collection. SMFPRMxx members are in the z/OS PARMLIB data set and can be modified to change the SMF recording interval and SMF types for the collection.

```

  Display  Filter  View  Print  Options  Search  Help
-----
SDSF OPERLOG  ZB01      05/04/2020    0W          43 RESP0
RESPONSE=ZB01
IEE967I 16.43.57 SMF PARAMETERS 978
          MEMBER = SMFPRM00
          NOWIC -- DEFAULT
          NOHFTSINTVL -- DEFAULT
          NOARECSIGN -- DEFAULT
          NORECSIGN -- DEFAULT
  
```

Figure 4-25 Displaying the current SMF PARMLIB member

Figure 4-26 shows the PARMLIB(SMFPRMxx) member.

```

BROWSE  SYSP.USER.PARMLIB(SMFPRMDV) - 01.02  Line 0000000000 Col 001 04
***** Top of Data *****
ACTIVE                                     /* REMOVE IEFUSI PYU, FEB 2000 */
RECORDING(LOGSTREAM)                     /* ACTIVE SMF RECORDING */
DEFAULTSNAME(IFASMF.DEFAULT)             /* RECORDING TO LOGSTREAM OR DATASET*/
LSNAME(IFASMF.STREAM.SYSTEM,TYPE(0:100,102:127)) /* ALL RECORDS WHICH ARE NOT
                                                    RECORDED ELSEWHERE (USER RECORDS)*/
LSNAME(IFASMF.STREAM.DB2,TYPE(101))      /* ALL SYSTEM RECORDS, EXCEPT DB2 RECORDS */
DPSIZMAX(1G)                             /* DB2 RECORDS */
DSNAME(SYSD.&SYSNAME..MAN1,
        SYSD.&SYSNAME..MAN2,
        SYSD.&SYSNAME..MAN3,
        SYSD.&SYSNAME..MAN4,
        SYSD.&SYSNAME..MAN5,
        SYSD.&SYSNAME..MAN6,
        SYSD.&SYSNAME..MAN7,
        SYSD.&SYSNAME..MAN8,
        SYSD.&SYSNAME..MAN9,
        SYSD.&SYSNAME..MANA)
INTVAL(05)
SYNOVAL(00)
NOPROMPT                                 /* PROMPT OPERATOR */
REC(PERM)                                /* TYPE 17 PERM RECORDS ONLY */
MAXDORM(3000)                            /* WRITE IDLE BUFFER AFTER 30 MIN */
STATUS(010000)                            /* WRITE SMF STATS AFTER 1 HOUR */
MEMLIMIT(16384P)
BUFSIZMAX(1G)                             /* MAXIMUM SIZE OF THE BUFFER */
JWT(2400)                                 /* 522 AFTER 24 HOURS */
SID(&SYSNAME.)
LISTON                                    /* LIST DATA SET STATUS AT IPL */
SYS(NOTYPE(16:19,62:63,65:69,92,99,100:102,110,119:120),
    EXITS(IEFU80,IEFU81,IEFACTRT,IEFU81,IEFU81,IEFU83),
    NOINTERVAL,NODETAIL)
SUBSYS(STC,EXITS(IEFUJI,IEFACTRT,IEFU83,IEFUSI))
SUBSYS(JES2,EXITS(IEFUJI,IEFACTRT,IEFU83,IEFUSI))
  
```

Figure 4-26 SMFPRMxxPARMLIB member

The example collects SMF data every 5 minutes (INTVAL(05)) across all SMF Types, except ranges 16 - 19, 62 - 63, 65 - 69, 99, 100 102, 110, and 119 - 120. Any changes to this member must be submitted in the SDSF log to account our changes: /SET SMF=xx.

Test that the definitions are correctly defined by running a query against the LOGSTREAM. Edit and run the following query in DVM Studio to get the results (see Figure 4-27).

```
SELECT * FROM "SMF_07001_LOG" LIMIT 1000;
```

	SMF_LEN	SMF_ZERO	SMF_FLAG	SMF_RTY	SMF_TIME	SMF_SID	SMF_SSI	SMF_STY	SMF_SEQN	SMF70RTY	SMF70TME	SMF70DTE	SMF70SID	SMF70SMF
0	25528	0	11011111	70	2020...	ZB02	RMF	1	...	70	3750000	2020308	ZB02	RMF
1	25528	0	11011111	70	2020...	ZB01	RMF	1	...	70	3750000	2020308	ZB01	RMF
2	25528	0	11011111	70	2020...	ZB02	RMF	1	...	70	3780000	2020308	ZB02	RMF
3	25528	0	11011111	70	2020...	ZB01	RMF	1	...	70	3780000	2020308	ZB01	RMF
4	25528	0	11011111	70	2020...	ZB02	RMF	1	...	70	3810000	2020308	ZB02	RMF
5	25528	0	11011111	70	2020...	ZB01	RMF	1	...	70	3810000	2020308	ZB01	RMF
6	25528	0	11011111	70	2020...	ZB01	RMF	1	...	70	3840000	2020308	ZB01	RMF
7	25528	0	11011111	70	2020...	ZB02	RMF	1	...	70	3840000	2020308	ZB02	RMF
8	25528	0	11011111	70	2020...	ZB02	RMF	1	...	70	3870000	2020308	ZB02	RMF

Figure 4-27 Result from SQL Query on SMF70 records with log stream

In-memory SMF access

Another way to read SMF data is to directly connect to in-memory buffers. This approach bypasses the SMF dump to GDG data set and SMF LOGSTREAM intervals. The method is similar to reading GDG or LOGSTREAM data sets.

In addition, we must modify the DVM configuration member AVZINS00 by adding the following statements after the GLOBAL PRODUCT OPTIONS statement:

```
IF DoThis
THEN DO
"DEFINE SMF NAME(IFASMF.INMEM)",
"STREAM(IFASMF.ZB01.INMEM)",
"BUFSIZE(500)",
"TIME(0)"
END
```

Pro tip: You must have your SMFPRMxx member in the system PARMLIB data set that is configured to use log streams and in-memory resources.

NAME is the name of the INMEMORY resource that matches the name of the resource that is defined to SMF with the INMEM parameter. If this parameter is included, the INMEMORY API is read continuously and a buffer of the most recent records is maintained. This parameter or the STREAM parameter, or both, must be specified. This parameter must begin with IFASMF.

Looking at the SMFPRMxx member in z/OS system PARMLIB in Figure 4-28, the INMEM parameter is an in-memory resource to record SMF records in memory for real-time processing.

```
BROWSE      SYSP.USER.PARMLIB(SMFPRMDV) - 01.03      Line 0000000000 Col 001 080
***** Top of Data *****
/* REMOVE IEFUSI      PYU, FEB 2000 */
ACTIVE          /* ACTIVE SMF RECORDING */
RECORDING(LOGSTREAM) /* RECORDING TO LOGSTREAM OR DATASET*/
DEFAULTSNAME(LOGSTREAM.&SYSNAME.) /* ALL RECORDS WHICH ARE NOT
RECORDED ELSEWHERE (USER RECORDS)*/
LSNAME(LOGSTREAM.&SYSNAME..SYSTEM,TYPE(0:100,102:127))
/* ALL SYSTEM RECORDS, EXCEPT DB2 RECORDS */
LSNAME(LOGSTREAM.&SYSNAME..DB2,TYPE(101)) /* DB2 RECORDS */
INMEM(IFASMF.&SYSNAME..INMEM,RESSIZMAX(1G),TYPE(30,70,72))
INMEM(IFASMF.&SYSNAME..INMEM.DB2,RESSIZMAX(1G),TYPE(101))
```

Figure 4-28 INMEM parameters in SMFPRMxxPARMLIB member

The INMEM parameter in SMFPRMxx features the following syntax:

```
INMEM(rname, RESSIZMAX({nnnnM|nG}), {TYPE({aa,bb|aa,bb:zz|aa,bb:zz,...})|
NOTYPE({aa,bb|aa,bb:zz|aa,bb:zz,...})}
```

The following subparameters are specified:

- ▶ rname is the name of the in-memory resource.
- ▶ RESSIZMAX defines the size of the buffer available for this in-memory resource, in megabytes or gigabytes.
- ▶ TYPE defines the SMF record types that are to be recorded to this in-memory resource.
- ▶ NOTYPE directs SMF is to collect all SMF record types, except record types that are specified.

Check the DVM server to ensure that in-memory log streams are updating the internal buffers (see Figure 4-29).

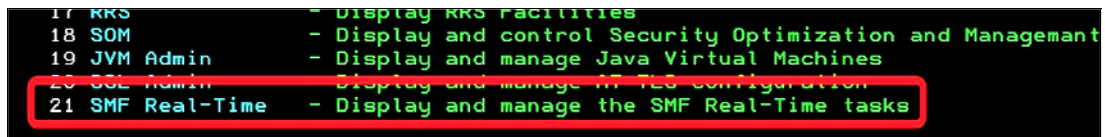


Figure 4-29 DVM server management menu

The internal buffer status for the in-memory resources that are defined earlier in SMFPRMxx is shown in Figure 4-30. The streams are enabled and active with available records.

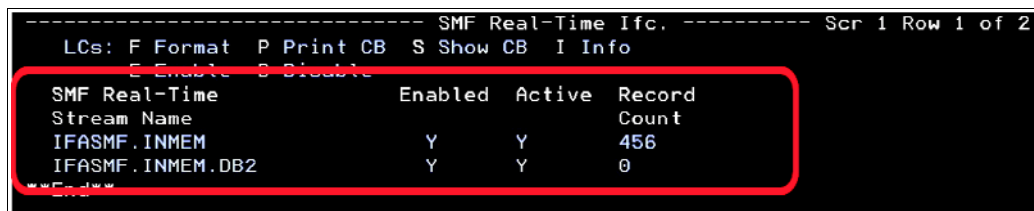


Figure 4-30 Displaying current SMF real-time streams

Use the following command to display SMF recording parameters and verify that in-memory streams are active (see Figure 4-31).

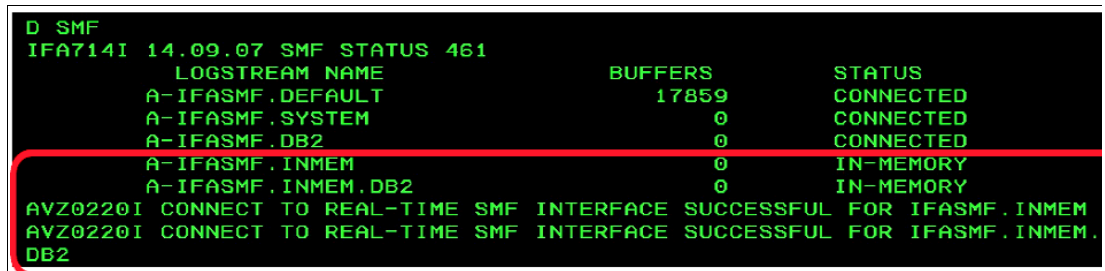


Figure 4-31 Displaying SMF recording parameters from LOGSTREAM

Similar to GDG or LOGSTREAM, enable rule AVZSMFT1 in the VTB rule set in the DVM server. In the global variables display, complete the following steps:

1. Change global prefix to GLOBAL2.
2. Select SMFTBL2 (see Figure 4-32).

```

----- Display Global Variables ----- Row 1 to 2 of 2
LCs: S Show Subnodes   M Modify Value   X Hex Browse
      D Remove Node and Subnodes       P Drop Node   B Browse

Global Prefix: GLOBAL2

S Subnode Name  Nodes  Subnode Value
-----
 8 SMFTBL2      13 NO VALUE ASSIGNED AT THIS LEVEL
   SYSLOG       2 NO VALUE ASSIGNED AT THIS LEVEL
**End**

```

Figure 4-32 Editing SMFTBL2 VTB rule

3. Configure the SMF data access option for IN-MEMORY by adding the following Global Variable to the existing variables:

```

S IM = "IFASMF.INMEM"
S IM2 = "IFASMF.INMEM.Db2"

```

You should see the new global variables in the list:

```

GLOBAL2.SMFTBL2.IM = "IFASMF.INMEM"
GLOBAL2.SMFTBL2.IM2 = "IFASMF.INMEM.Db2"

```

Pro tip: Be careful when defining the data set name. You must enter the name in uppercase characters. If you specify the correct name, but in lower or mixed cases, the allocation fails.

4. Submit the following SQL Query to display SMF30 records (which are collected), from the IFASMF.INMEM buffer (see Figure 4-33) displays records that are captured in real-time through in-memory SMF collection. The SQL syntax for Virtual Tables is composed of the Table Mapping (SMF_03000) with IM (IFASMF.INMEM) separated by a 'single' underscore "_":

```
SELECT * FROM SMF_03000_IM LIMIT 10000;
```

	SMF_LEN	SMF_ZERO	SMF_FLAG	SMF_RTY	SMF_TIME	SMF_SID	SMF_SSI	SMF_STY	SMF_SEQN	SMF30SOF	SMF30SLN	SM ^
72	1422	0	11011110	30	2020-05-20 14:31:53.37	ZB01	OMVS	4	...	192	38	1
73	1422	0	11011110	30	2020-05-20 14:31:53.37	ZB01	OMVS	5	...	192	38	1
74	1422	0	11011110	30	2020-05-20 14:31:53.38	ZB01	OMVS	4	...	192	38	1
75	1422	0	11011110	30	2020-05-20 14:31:53.38	ZB01	OMVS	5	...	192	38	1
76	1422	0	11011110	30	2020-05-20 14:31:53.38	ZB01	OMVS	4	...	192	38	1

Figure 4-33 Result from SQL Query on SMF30 in-memory records

How SMF records are mapped in DVM for z/OS

Although Data Studio makes it easy to access fields in SMF records, you still need to understand the underlying data and how it is structured. SMF records typically vary in length and allow the same record type to contain various amounts of information.

This mechanism is a flexible and effective way to present the maximum amount of information in the smallest amount of space. Maintain relationships between the rows in the various tables by adding a field to the table that contains the base part of the record and a corresponding field in the tables that contains the repeating sections.

DVM Studio shows nearly every record type that is mapped. One type is called SMF_tttss, and one or more is called SMF_tttss_aaaaaa, where ttt is the record type (in decimal), ss is the subtype, and aaaaaa is a string of characters and numbers that indicate the repeating section that resides in that table.

For example, the type 70 subtype 1 record is loaded into the following tables:

- ▶ SMF_07001
- ▶ SMF_07001_SMF70AID
- ▶ SMF_07001_SMF70BCT

In DVM terminology, the table that contains the record header is called the *base table* and the tables that contain the repeating sections are called *subtables*. The base table contains a generated column that is called CHILD_KEY, and the subtables contain a generated field that is called PARENT_KEY. All of the rows in the base table and the subtables that include the same CHILD_KEY and PARENT_KEY values came from the same SMF record.

If you want to extract fields from the base table and from repeating sections that were in the same record, you run a JOIN between the PARENT_KEY and CHILD_KEY fields, as shown in the following example:

```
SELECT SMF_TIME, SMF_SID, SMF_SEQN, SMF70VPA, SMF70BPS,  
FROM SMF_07001 A0 JOIN SMF_07001_SMF70BPD A9  
ON A0.CHILD_KEY = A9.PARENT_KEY;
```

Supported SMF record types

Rocket is constantly developing support for more SMF record types and shipping PTFs to deliver that support to customers. If you are trying to determine if a record type is supported, the easiest way to make that determination is to scroll through the list of support virtual tables in DVM Studio.

You can also get a list of the record types and the field names (but not the subtable names) from the ISPF interface (see Figure 4-34) from the primary DVM server menu.



Figure 4-34 Displaying maps in the DVM server

A row for each record type and subtype is available (see Figure 4-35).

SMF_0630	Enabled	Y	Sequential	**/**/**	00:00	TSELP	
SMF_0630	Enabled	Y	View	**/**/**	00:00	TSELP	
SMF_0640	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	
SMF_0650	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	
SMF_0660	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	
X SMF_0700	Enabled	Y	Sequential	**/**/**	00:00	TS6242	DISPLAYED
SMF_0710	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	
SMF_0720	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	
SMF_0720	Enabled	Y	Sequential	**/**/**	00:00	TS0SB	

Figure 4-35 SMF records mappings

Enter an X next to the base table and a list of all the fields in that base table and all of its subtables, as shown in Figure 4-36.

```

----- Data Mapping Elements ----- Scr 1 Row 1 of 426
LCs: P Print Map S Show Map D Disable E Enable C Change
Map name: SMF_07001
FIELD          COLUMN
NAME           NAME           NOTE
SMFREC         SMFREC
SMF70HDR       SMF70HDR
SMF_LEN        SMF_LEN
SMF_ZERO       SMF_ZERO
SMF_FLAG       SMF_FLAG
  
```

Figure 4-36 Displaying MAP for SMF70 records

By scrolling to the right, you can also see the definition of each field, such as its format, length, and offset (see Figure 4-37).

```

----- Data Mapping Elements ----- Scr 2 Row 1 of 426
LCs: P Print Map S Show Map D Disable E Enable C Change
Map name: SMF_07001
FIELD          LEVEL LENGTH FORMAT      OFFSET STATUS SEQ
NAME           NAME           NAME
SMFREC         1      6904  Group      0      Disabled 1
SMF70HDR       2      198   Group      0      Disabled 2
SMF_LEN        3       4     Int        0      Enabled  3
SMF_ZERO       3       4     Int        4      Enabled  4
SMF_FLAG       3       8     Character  8      Enabled  5
SMF_DTV        2       2     Small Int 16     Enabled  6
  
```

Figure 4-37 Displaying SMF70 records mapping definitions

4.6.5 Db2 unload data sets

To access a Db2 unload data set directly with an SQL query, you must configure a virtual table rule to define the Db2 unload data set name to the Db2 virtual table.

To configure access to a Db2 unload data set, you must add the Db2 unload data set name to the Db2 virtual table in a Data Virtualization Manager Server Event Facility (SEF) virtual table rule. With this access, you can issue SQL queries directly against Db2 unload data sets for existing Db2 virtual tables.

Switching a Db2 virtual table to read an unload data set is done by assigning a data set name to the table in a virtual table rule. The VTB global variable `vtb.optbdsna` is used to redirect access from Db2 to reading the sequential file that is named in the variable. The named sequential file must contain the unload data that is created by the Db2 UNLOAD utility. A model VTB rule, AVZMDLDU, is provided to demonstrate redirecting a Db2 virtual table to a Db2 unload data set.

For the example that is shown in Figure 4-38, consider a virtual table that is named DW01_DSN81210_EMP that maps to the EMP table in the Db2 subsystem DW01.

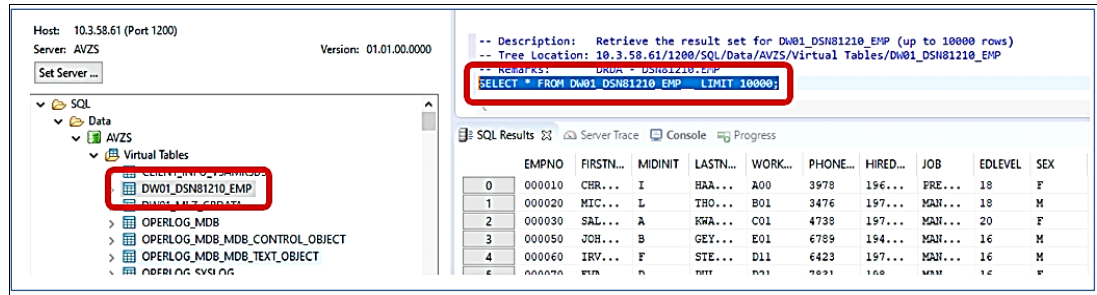


Figure 4-38 DW01_DSN81210_EMP virtual table

By activating the model rule AVZMDLDU, you can query an unloaded sequential data set named DSNDW00_UNLD_DSN81210_EMP by issuing the following query. The results are shown in Figure 4-39:

```
SELECT * FROM MDLDU_DW01_DSN81210_EMP_ARNOULD_DW00_UNLD_DSN8D12A_DSN8S12E;
```

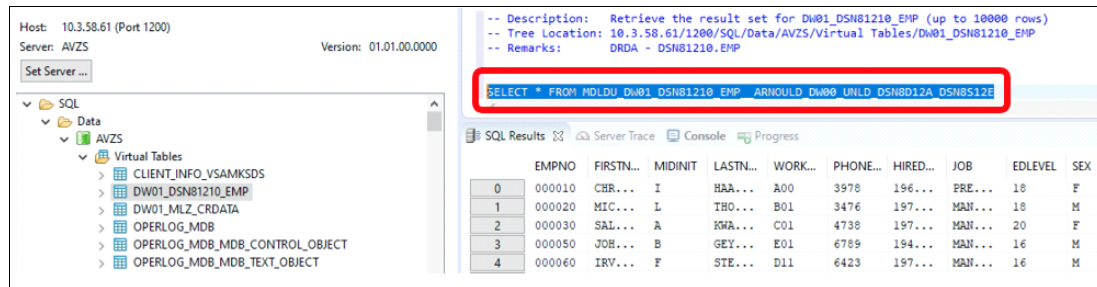


Figure 4-39 Result from SQL Query access to an UNLOAD data set for DW01_DSN81210_EMP

The AVZMDLDU rule performs the following steps:

1. Extracts the table name DW01_DSN81210_EMP and sets the VTB global variable vtb.optbmtna.
2. Extracts the data set name ARNOULD_DW00_UNLD_DSN8D12A_DSN8S12E, converts the underscores to periods, and then, sets the VTB global variable vtb.optbdsna.

The following restrictions and considerations apply for this feature:

- ▶ SQL access to Db2 unload files is limited to SQL queries only.
- ▶ The columns in the Db2 virtual table definition must exactly match the table that is unloaded in Db2.
- ▶ To use this feature, the corresponding Virtual Table must exist in the DVM server.

The sample rule AVZMDLDU can be used as a reference for more customization. When customizing this rule, more logic is likely needed to be added if different unload data sets require different VTB variable settings for CCSID or internal or external format.

Customize the Data Virtualization Manager configuration member (AVZSIN00) to enable virtual table rule events by configuring the SEFVTBEVENTS parameter in the member (see Figure 4-40).

```

BROWSE      DVM.ZB01.AVZS.SAVZEXEC(AVZSIN00) - 01.46 Line 0000000254 Col
/*-----*/
/* Enable Server Event Facility support (SEF). Issue a DEFINE          */
/* RULESET statement for each SEF ruleset.                            */
/*                                                                      */
/* During initial installation you must change each DSNAME operand   */
/* to match the data set names actually installed with the product.  */
/* SWICNTLDSN is used for HTTP calls issued by the Studio.          */
/*-----*/
"MODIFY PARM NAME (SEFMAXSECONDS) VALUE (120) "
"MODIFY PARM NAME (SEFVTBEVENTS) VALUE (YES) "

```

Figure 4-40 Activating SEFVTBEVENTS in AVZSIN00 member

Complete the following steps to access the VTB rules:

1. In the Data Virtualization Manager server Primary Option Menu, specify option E, Rules Mgmt.
2. Specify option 2, SEF Rule Management.
3. Enter VTB for Display Only the Ruleset Named setting (see Figure 4-41).

```

----- Event Facility (SEF) Ruleset Entry Profile ----- SSID: AVZS

Display Only the Ruleset Named   ==> VTB _____ ( * = display all rulesets )
Limit Display to These Rule Types ==> * _____ ( * = display all types )

Require Current Member Statistics ==> Y _____ ( N = bypass directory reads )
Confirm Mass Changes             ==> Y _____ ( N = bypass confirmations )
Display Entry Selection Panel    ==> Y _____ ( N = bypass entry panel )

```

Figure 4-41 Displaying VTB rules only

4. Complete the following steps to customize the AVZMDLDU rule:
 - a. Specify S next to AVZMDLDU to edit the rule (see Figure 4-42).

```

----- Event Facility (SEF) Event Procedure List Row 9 to 22 of 22
LCs: S ISPF Edit E Enable D Disable A Set Auto-Enable
      Z Reset Auto-Enable B Set Auto/Enable C Disable/Reset Auto

PDS Members for: DVM.ZB01.AVZS.SAVZXVTB
      A
S Member Status E TYP VV.MM Created Modified Size Init Mod ID
-----
S AVZMDLDU ENABLED Y VTB 01.11 20/04/28 20/05/15 15:06 76 76 5 ARNOULD
AVZMDLDU ENABLED Y VTB 01.00 20/04/20 20/04/20 12:11 60 60 0 ARNOULD

```

Figure 4-42 Editing VTB rule AVZMDLDU

- b. Find the `vtb.optbdsna` variable and specify the name of the Db2 unload data set to process (see Figure 4-43).

```

/*-----*/
/* Remove the "MDLDU_" prefix to get the DMF map name */
/* and unload dataset name */
/*-----1-----2-----3-----4-----5-----6-----7*/
parse var vtb.optbtbna Pref "_" DB2Tab "_" UnloadDsn
if UnloadDsn <> "" then do
  vtb.optbmtna = DB2Tab /* remove MDLDU_ from table name */
  UnloadDsn = TRANSLATE(UnloadDsn,".", "_") /* convert _ to . */
/*
vtb.optbdsna = UnloadDsn set the dataset name to process */
vtb.optbdsna = DRDA - DSN81210 - DSN8120

```

Figure 4-43 `vtb.optbdsna` variable in AVZMDLDU rule

Pro Tip: If you update the sample AVZMDLDU rule as described in the preceding steps, you can provide a default value for the `vtb.optbdsna` variable. By updating this rule, you do not need to specify the correct dataset name in the SQL statement for execution. Figure 4-44 shows a simple SQL that returns values from the default UNLOAD dataset.

- c. Run the following SELECT command to query the UNLOAD data set:
- ```
SELECT * FROM MDLDU_DW01_DSN81210_EMP_WHATEVER_DATASET;
```

Figure 4-44 Result from SQL Query specifying UNLOAD Dataset name

If you do not want to see this behavior, you do not need to modify the sample rule `vtb.optbdsna` variable. Every SQL must specify the correct UNLOAD data set name to work correctly.

- d. Update rule options as needed. Figure 4-45 shows the VTB rule options that support Db2 unload data set access.

```

/*-----*/
/* If the data was unloaded in internal format using a */
/* FORMAT INTERNAL statement, set vtb.optbduif to 1. */
/*-----1-----2-----3-----4-----5-----6-----7*/
/* vtb.optbduif = 0 /* Format is external format */
/*-----*/
/* If the data was unloaded in delimited format using a */
/* DELIMITED statement, set vtb.optbdlcv to 1. */
/*-----1-----2-----3-----4-----5-----6-----7*/
/* vtb.optbdlcv = 1 /* Format is not delimited */
/*-----*/
/* If the CCSID is not compatible with the SQL engine default */
/* CCSID, set vtb.optbtbcc. */
/*-----1-----2-----3-----4-----5-----6-----7*/
/* vtb.optbtbcc = 1140 /* Use default engine CCSID */
/*-----*/
end

```

Figure 4-45 Options for AVZMDLDU rule

The following list has the VTB variables are available

- vtb.optbdlcv: If the data was unloaded with a DELIMITED statement, set vtb.optbdlcv to 1 to declare the data is in delimited format. It might be necessary to declare the delimiters if the default column delimiter (,) and character string delimiter (“) were overridden when the data was unloaded.
  - vtb.optbdsna: Specifies the name of the sequential unload data set that was created by the Db2 UNLOAD utility to access.
  - vtb.optbduif: By default, the Db2 unload utility writes data in external format. If FORMAT INTERNAL is used when unloading data, vtb.optbduif must be set to 1 to declare that the data was unloaded in internal format.
  - vtb.optbmtna: Specifies the map name of the Db2 virtual table that is describing the unload file.
  - vtb.optbtbcc: Is used if the table CCSID is not compatible with the CCSID defined for the SQL engine (AVZSIN00 SLENGLFTCCSID parameter).
  - vtb.optbtbcc: Can be used to declare the CCSID of the data. It is important for Unicode tables and tables that contain GRAPHIC columns.
5. Enable the rule by specifying E next to AVZMDLDU and then, press **Enter**.
  6. Set the rule to auto-enable by specifying A next to AVZMDLDU, as shown in Figure 4-46. Then, press **Enter**. Setting a rule to Auto-enable activates the rule automatically when the server is restarted.

```

----- Event Facility (SEF) Event Procedure List Row 9 to 22 of 22
LCs: S ISPF Edit E Enable D Disable A Set Auto-Enable
 Z Reset Auto-Enable B Set Auto/Enable C Disable/Reset Auto

PDS Members for: DVM.ZB01.AVZS.SAVZXVTB
 A
S Member Status E TYP VV.MM Created Modified Size Init Mod ID

A AVZMDLDU ENABLED Y VTB 01.11 20/04/28 20/05/15 15:06 76 76 5 ARNOULD
AVZMDLVC ENABLED Y VTB 01.00 20/04/28 20/04/28 10:11 60 60 0 ARNOULD

```

Figure 4-46 Auto-enabling the AVZMDLDU VTB rule



## Connecting to non-Z data sources

Many organizations must integrate structured data from various relational database management systems (RDBMS) that are not on the mainframe environment, such as Linux, UNIX, Windows, and the cloud.

Data Virtualization Manager for z/OS (DVM for z/OS) can access relational databases by using built-in DRDA connections. DVM for z/OS also provides an application server (the JDBC Gateway Server) that provides access to many non-mainframe JDBC data sources.

In this chapter, we discuss how to connect with non-mainframe data sources by using the JDBC Gateway server client component.

This chapter includes the following topics:

- ▶ 5.1, “Introduction” on page 80
- ▶ 5.2, “Accessing non-z/OS data sources by using the JDBC Gateway server” on page 81

## 5.1 Introduction

Many enterprises that maintain business-critical data on the mainframe also need to integrate structured data from various RDBMS systems that are off of the mainframe environment (distributed, Linux, UNIX, and Windows-based). Several distributed data sources can be accessed by using built-in DRDA connections (as described in 1.2, “Why DVM for z/OS in modernization?” on page 2). DVM for z/OS also provides an application server (the JDBC Gateway server) that provides access to many non-mainframe JDBC data sources, as shown in Figure 5-1.

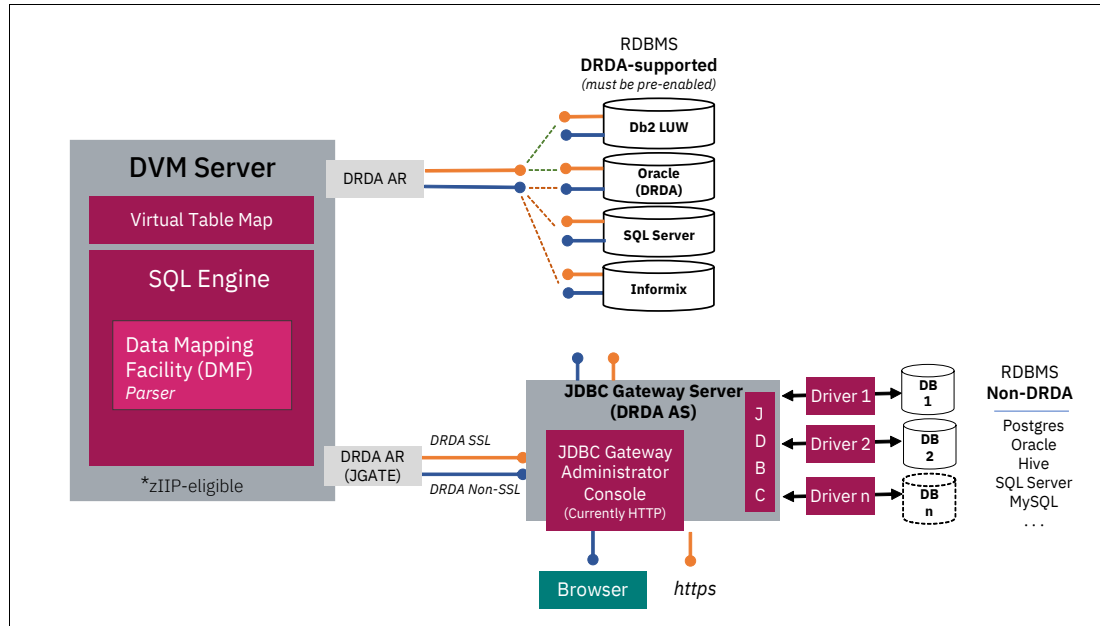


Figure 5-1 Example of non-z/OS endpoint support

This chapter references the configurations and processes that are specific to different distributed databases, such as IBM DB2 Big SQL, IBM Db2 Warehouse, on-premises Db2 Family, Microsoft SQL Server, and many other data sources that use the JDBC Gateway server.

### 5.1.1 Standard access to data sources

Standard access to data sources is performed by client requesters that use ODBC or JDBC compliant connections. These protocols connect to the DVM server and issue access requests to specific data sources with accompanying DML operations to be performed.

### 5.1.2 Distributed Relational Database Architecture

Many distributed databases can also be accessed by configuring standard Distributed Relational Database Architecture (DRDA) access methods.

To access distributed databases, the DVM server must be configured to use the **DEFINE DATABASE** definition in the DVM configuration server member `h1q.xVZy.SAVZEXEC.(xVZyIN00)`. After it is added, the DVM must be restarted.

For IBM Db2 family products that are running with a built-in federation capability, you can use nicknames and remote tables to connect to the DVM server to access mainframe data sources. The Db2 distributed portfolio, including Data Warehouse offerings, is shipped with the DVM server ODBC/JDBC drivers, and are set up and configured, ready for use.

Connecting to non-z/OS sources through DRDA, such as Oracle, the Oracle Database Provider for DRDA is required. A Host Integration Server (HIS) DRDA Service is required for connections to a Microsoft SQL Server database. For more information, see the Microsoft documentation [Configuring Service for DRDA](#).

Generally, the configuration member needs updating with more configuration for the DVM server Event Facility (SEF) rules, as needed. Optionally, any alternative authentication information also can be configured. Within the DVM configuration member, specify the DRDA RDBMS settings through a definition statement and provide local environment values for all the parameters. As shown in the Define Database specification in Example 5-1, the DVM Started Task must be recycled.

*Example 5-1 DVM configuration member definition*

---

```
"DEFINE DATABASE TYPE(type_selection)",
"NAME(name)",
"LOCATION(location)",
"DDFSTATUS(ENABLE)",
"DOMAIN(your.domain.name)",
"PORT(port)",
"IPADDR(1.1.1.1)",
"CCSID(37)",
"APPLNAME(DSN1LU)",
"IDLETIME(110)"
```

---

For more information, see [IBM Documentation](#).

## 5.2 Accessing non-z/OS data sources by using the JDBC Gateway server

In this section, we demonstrate the installation and configuration process for the JDBC Gateway server and provide an example for accessing a remote data source. In our overview, we refer to the following components that are involved in the installation and configuration of the JDBC Gateway server environment:

- ▶ JDBC Gateway server (JGATE) is the server that is the back-end component that bridges the communication between the DVM server and remote data source.
- ▶ JDBC Gateway Admin UI is the administrative console that serves as the front-end web user interface that is used to configure data sources.
- ▶ DVM server is the resident server that is running in a z/OS started task that receives the incoming connections, virtualizes data sources, and processes the SQL requests against the data source.

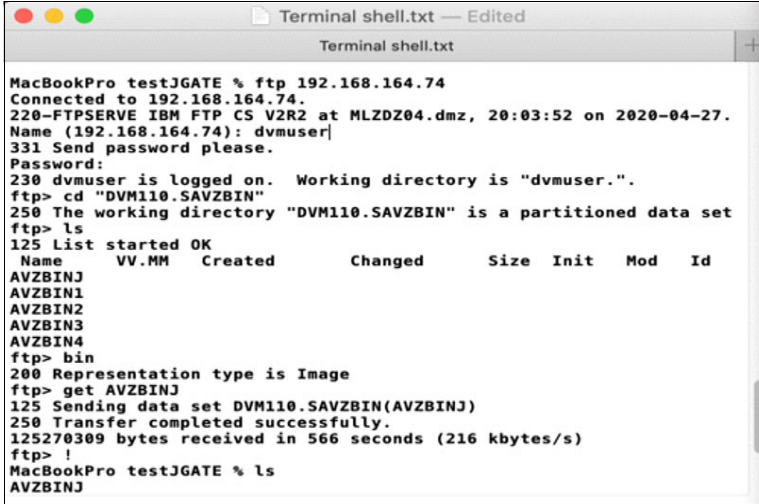
## 5.2.1 Setting up the JDBC Gateway server

In this section, we review how to prepare for installing the JDBC Gateway server. IBM Documentation provides more information about the JDBC Gateway server installation process at [IBM Documentation](#).

The JDBC Gateway server is a pure Java application, which can run on any platform that supports Java 8 or higher.

**Note:** If it is not practical to install the JDBC Gateway server on the target data source's platform, it can also be installed in a UNIX System Services environment on the mainframe.

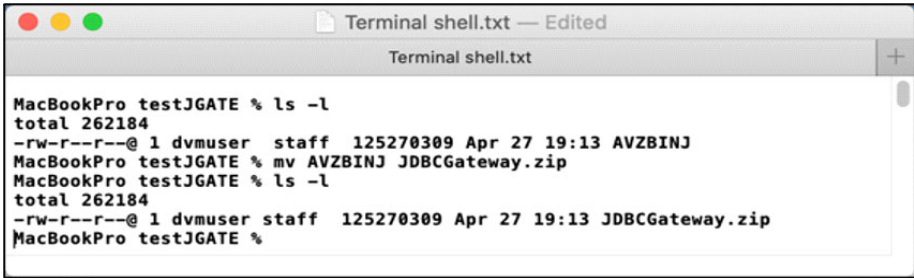
The installation file can be obtained from the IBM Support Fix Central [download site](#). This file can be transferred from the mainframe that uses FTP, renamed, and unarchived. The resulting JAR file can be run on the target platform to complete the server installation, as shown in Figure 5-2.



```
Terminal shell.txt — Edited
Terminal shell.txt
MacBookPro testJGATE % ftp 192.168.164.74
Connected to 192.168.164.74.
220-FTPSERVE IBM FTP CS V2R2 at MLZDZ04.dmz, 20:03:52 on 2020-04-27.
Name (192.168.164.74): dvmuser|
331 Send password please.
Password:
230 dvmuser is logged on. Working directory is "dvmuser.".
ftp> cd "DVM110.SAVZBIN"
250 The working directory "DVM110.SAVZBIN" is a partitioned data set
ftp> ls
125 List started OK
 Name VV.MM Created Changed Size Init Mod Id
AVZBINJ
AVZBIN1
AVZBIN2
AVZBIN3
AVZBIN4
ftp> bin
200 Representation type is Image
ftp> get AVZBINJ
125 Sending data set DVM110.SAVZBIN(AVZBINJ)
250 Transfer completed successfully.
125270309 bytes received in 566 seconds (216 kbytes/s)
ftp> !
MacBookPro testJGATE % ls
AVZBINJ
```

Figure 5-2 FTP transfer of the JGATE server installation file

After the installation file is downloaded, it can be renamed to JDBCGateway.zip and transferred (for example, by using FTP) to the target platform to be installed, as shown in Figure 5-3.

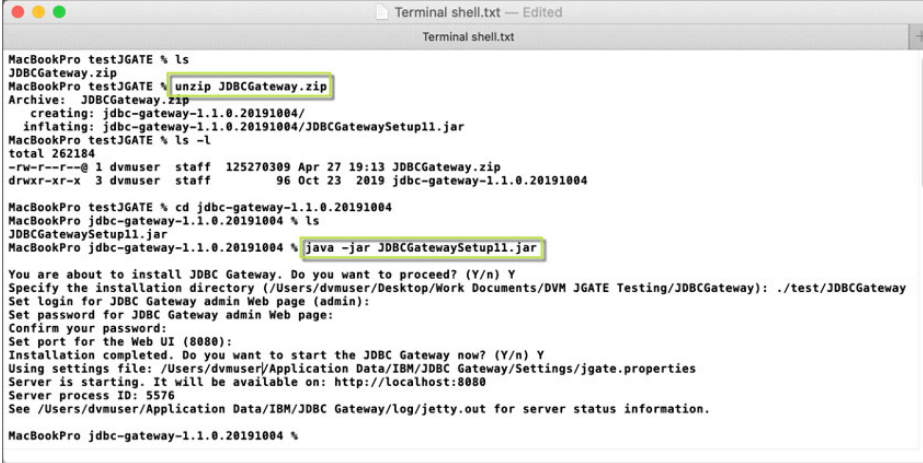


```
Terminal shell.txt — Edited
Terminal shell.txt
MacBookPro testJGATE % ls -l
total 262184
-rw-r--r--@ 1 dvmuser staff 125270309 Apr 27 19:13 AVZBINJ
MacBookPro testJGATE % mv AVZBINJ JDBCGateway.zip
MacBookPro testJGATE % ls -l
total 262184
-rw-r--r--@ 1 dvmuser staff 125270309 Apr 27 19:13 JDBCGateway.zip
MacBookPro testJGATE %
```

Figure 5-3 - Renaming J GATE installation file

## 5.2.2 Installing the JDBC Gateway server

Figure 5-4 shows the process for extracting and installing the JDBC Gateway server client component.



```
MacBookPro testJGATE % ls
JDBCGateway.zip
MacBookPro testJGATE % unzip JDBCGateway.zip
Archive: JDBCGateway.zip
 creating: jdbc-gateway-1.1.0.20191004/
 inflating: jdbc-gateway-1.1.0.20191004/JDBCGatewaySetup11.jar
MacBookPro testJGATE % ls -l
total 262184
-rw-r--r--@ 1 dvmuser staff 125270309 Apr 27 19:13 JDBCGateway.zip
drwxr-xr-x 3 dvmuser staff 96 Oct 23 2019 jdbc-gateway-1.1.0.20191004

MacBookPro testJGATE % cd jdbc-gateway-1.1.0.20191004
MacBookPro jdbc-gateway-1.1.0.20191004 % ls
JDBCGatewaySetup11.jar
MacBookPro jdbc-gateway-1.1.0.20191004 % java -jar JDBCGatewaySetup11.jar

You are about to install JDBC Gateway. Do you want to proceed? (Y/n) Y
Specify the installation directory (/Users/dvmuser/Desktop/Work Documents/DVM JGATE Testing/JDBCGateway): ./test/JDBCGateway
Set login for JDBC Gateway admin Web page (admin):
Set password for JDBC Gateway admin Web page:
Confirm your password:
Set port for the Web UI (8080):
Installation completed. Do you want to start the JDBC Gateway now? (Y/n) Y
Using settings file: /Users/dvmuser/Application Data/IBM/JDBC Gateway/Settings/jgate.properties
Server is starting. It will be available on: http://localhost:8080
Server process ID: 5576
See /Users/dvmuser/Application Data/IBM/JDBC Gateway/Log/jetty.out for server status information.

MacBookPro jdbc-gateway-1.1.0.20191004 %
```

Figure 5-4 JGATE server installation

Extract the JDBCGateway.zip in the library where it was transferred to the target system. If your host machine does not have an extract utility, extract the contents of the installation file on a Windows workstation and copy the JDBCGatewaySetup11.jar file to the host machine.

Change to the directory where the installation package was extracted:

```
cd jdbc-gateway-x.x.x.xxxxxxx
```

Run the following command:

```
java -jar JDBCGatewaySetup11.jar
```

## 5.2.3 Running JDBC Gateway server by using UNIX System Services

The JDBC Gateway server also can be installed in a UNIX System Services (OMVS) environment on the mainframe. Information about this process, including configuration settings that are required for a UNIX System Services installation, is described in the IBM Documentation article, [Installing JDBC Gateway](#).

The UNIX System Services environment provides a centralized location for the JDBC Gateway server to access remote data sources that are provisioned by the DVM server. Isolating the JDBC Gateway server from the DVM server results in better co-location with source data. Because the JDBC Gateway server is a pure Java application, it can run anywhere that Java 8 is supported.

### Configuration

For installation in UNIX System Services, it is recommended that you define the following environment variables:

- ▶ export IBM\_JAVA\_OPTIONS="-Dfile.encoding=ISO8859-1"
- ▶ export \_BPXX\_AUTOCVT=ON

When the JDBC Gateway server installer generates Start and Stop scripts, the following actions occur (depending on these variables):

- ▶ If the recommended environment variables are not set, the scripts are generated in EBCDIC. You can run the gateway as normal for UNIX by using the **sh startServer.sh** command.
- ▶ If you set the `IBM_JAVA_OPTIONS` variable, the scripts are generated in ASCII, and you must use the **ctag -tc IS08859-1** command.

Tagging in UNIX System Services means that `_BPXX_AUTOCVT` must be `ON` if you want to edit or run the script in the shell.

- ▶ Files that are generated by the JDBC Gateway server, such as log files and the `jgate.properties` file, are generated in ASCII regardless of the environment variable settings (the exception is `jetty.out`, which is in EBCDIC). To browse these files in UNIX System Services, you must use the `ctag` command and set `_BPXX_AUTOCVT=ON`.

## Customized installation

As a best practice, install the JDBC Gateway server to a non-user library in UNIX System Services. Doing so allows centralized access to the Gateway server by any authorized user. It also prevents issues where an administrator user's access permissions change or are no longer accessible to the platform.

When running the `JDBCGatewaySetup11.jar` executable, override the `Duser.home` variable by using the `Duser.home` argument and point it to a directory. This directory must be empty before proceeding with this type of installation:

```
java -Duser.home=<dir where the properties and config files are created> -jar
JDBCGatewaySetup11.jar
```

```
java -Duser.home="C:\ZDVM\product\JGateway3-config" -jar JDBCGatewaySetup11.jar
```

After successful installation, the `startServer.cmd` and `stopServer.cmd` must be updated. When the `startServer.sh` is run, the new directory path is used:

```
java -Duser.home=<dir where the properties and config files were created> -jar
JDBCGatewaySetup11.jar
```

```
java -Duser.home="C:\ZDVM\product\JGateway3-config" -jar JDBCGateway.jar
```



## Starting the JDBC Gateway server to use batch execution

Use a batch interface as a best practice to start and stop the JDBC Gateway server in a UNIX System Services environment. The JCL examples that are shown in Figure 5-5 can be used to start the `startServer.sh` and `stopServer.sh` scripts.

```
START Server
***** Top of Data *****
//JGSTART JOB MSGLEVEL(1,1),
// MSGCLASS=A,
// USER=ABCDEFG,
// NOTIFY=ABCDEFG
//*****
//* JGATE Server *
//*****
//DVMINVOK EXEC PGM=BPXBATCH,REGION=0M,
// PARM='SH /abcdefg/jgate/JDBCGateway/startServer.sh'
//STDOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
***** Bottom of Data *****

STOP Server
***** Top of Data *****
//JGSTOP JOB MSGLEVEL(1,1),
// MSGCLASS=A,
// USER=ABCDEFG,
// NOTIFY=ABCDEFG
//*****
//* JGATE Server *
//*****
//JGAINVOK EXEC PGM=BPXBATCH,REGION=0M,
// PARM='SH /abcdefg/jgate/JDBCGateway/stopServer.sh'
//STDOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
***** Bottom of Data *****
```

Figure 5-5 The use of batch scripting to efficiently start and stop the JDBC Gateway server

The use of the start and stop scripts directly in OMVS limits the number of resources that is available to the JDBC Gateway server for a user's OMVS segment. Starting from JCL with BPXBATCH, the customer can set the REGION size, as shown in Figure 5-5.

### 5.2.4 Managing JDBC Gateway server software upgrades

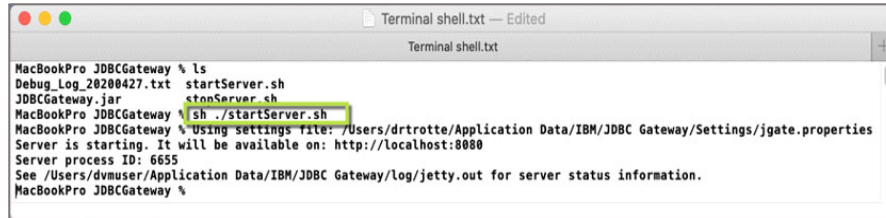
Periodically, the JDBC Gateway server must apply fix levels or new releases. To ensure that all of the current environment and configuration settings are preserved, a backup of the `startServer.sh` and `stopServer.sh` scripts are required. After the new fix or release level of the software is downloaded, install the newer version by using the following command:

```
java -jar JDBCGatewaySetupxx.jar
```

The installation script automatically installs by using the original installation directory. Reply Y when prompted to clear the directory and edit the new `startServer.sh` and `stopServer.sh` scripts to re-implement any changes from the previous installation.

## 5.2.5 Starting the JDBC Gateway server that uses administrative UI

The Gateway server also can be used as an administrative UI. The JDBC Gateway server can be started and stopped on an MS-Windows, Linux, UNIX, Mac OSx, and web browser (see Figure 5-6).



```
MacBookPro JDBCGateway % ls
Debug_Log_20200427.txt startServer.sh
JDBCGateway.jar stopServer.sh
MacBookPro JDBCGateway % sh ./startServer.sh
MacBookPro JDBCGateway % Using settings file: /Users/drtrotte/Application Data/IBM/JDBC Gateway/Settings/jgate.properties
Server is starting. It will be available on: http://localhost:8080
Server process ID: 6655
See /Users/dvmuser/Application Data/IBM/JDBC Gateway/log/jetty.out for server status information.
MacBookPro JDBCGateway %
```

Figure 5-6 Starting JGATE server

At a command prompt in the JDBC Gateway server installation directory, run one of the following commands to start or stop the Gateway server on MS-Windows or Linux, Windows, or Mac OSx:

- ▶ MS-Windows: startServer, stopServer
- ▶ Linux, UNIX, or Mac OSx: sh startServer.sh, sh stopServer.sh

A web browser can be used by connecting to the server IP address and port that were chosen for the Admin UI during installation (that is, <https://192.168.1.31:8091>), in which a username and password prompt appears. The default username is admin.

The user interface initializes and a user access dialog window appears in which you are prompted to enter a username (the default is admin) and a password that you chose during the server installation process, as shown in Figure 5-7.

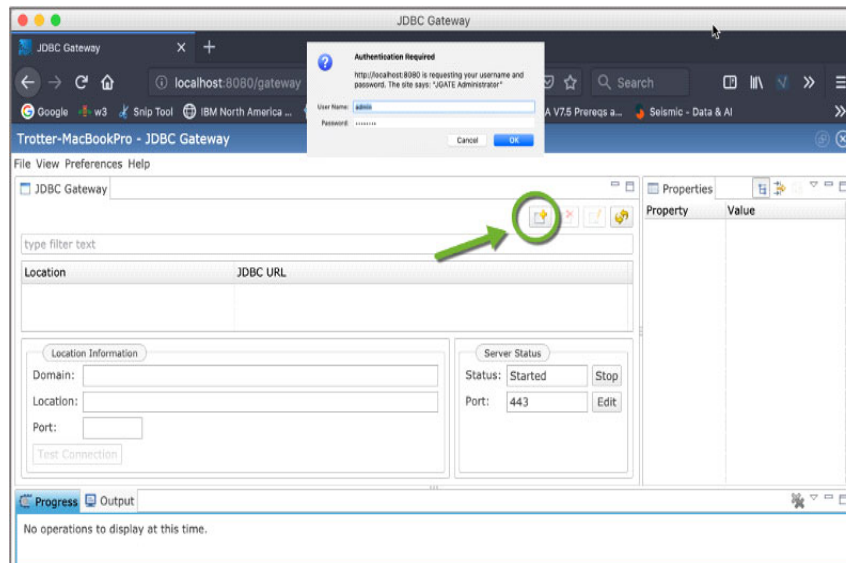


Figure 5-7 New Data Source configuration window

## 5.2.6 Configuring data sources that use the JDBC Gateway server UI

Data sources can now be configured by using the administration UI console. For our example, we add a connection to a PostgreSQL database on a remote server (see Figure 5-8).



Figure 5-8 Adding a data source

By using the Connections parameters drop-down list for the JDBC, supported data sources can be selected. If the data source does not exist, the ellipsis can be selected for a new data source, as shown in Figure 5-9.

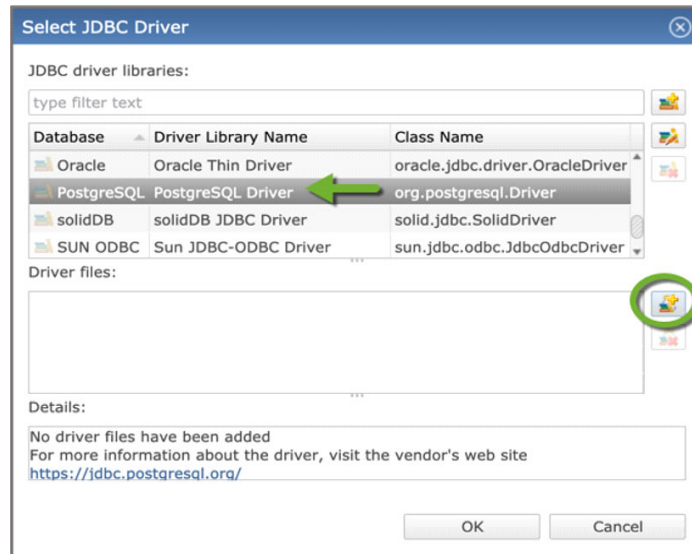


Figure 5-9 Selecting JDBC driver

After the data source is selected, JDBC connection string details can be added for the target database that uses the suitable format, as shown in Figure 5-10.

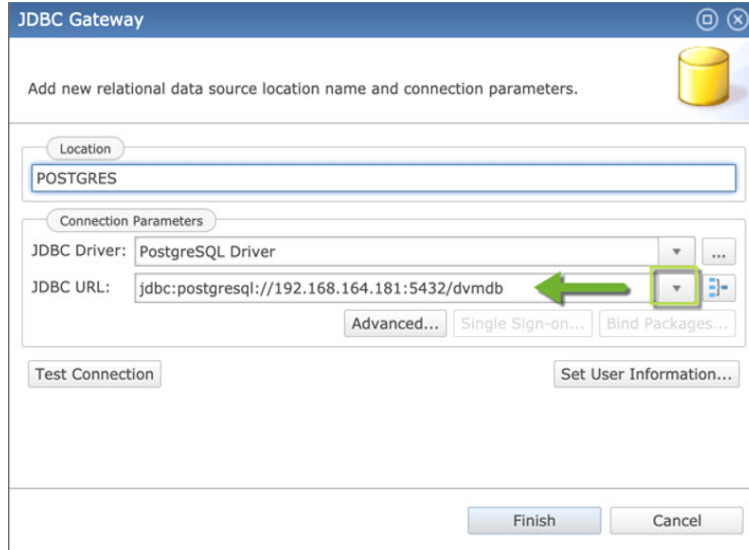


Figure 5-10 Data source connection details

Access and authentication credentials also are maintained for access to the target database with the ability to test a successful connection, as shown in Figure 5-11.

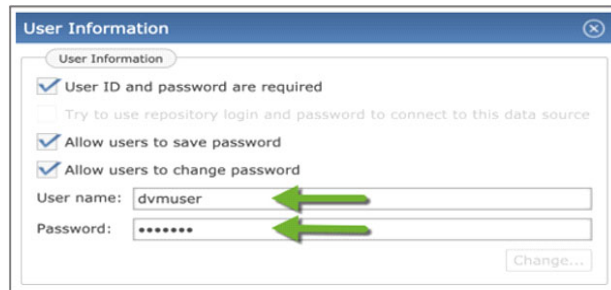


Figure 5-11 Define user access for a target data source

## 5.2.7 Configuring the DVM server to access the JDBC Gateway server

To create a mapping or linkage between the requesting DVM server and a newly installed JDBC Gateway server, a z/OS TSO/e session is needed to update DVM server configuration members. Depending on the SMP/e installation and the naming that is chosen for the DVM server subsystem, the server initialization member is found by finding the h1q.xVzy.SAVZEXEC(xVzyIN00) configuration libraries.

By using ISPF, the installation library can be found in DVM110.AVZ1.SAVZEXEC, where the xVZyIN00 configuration member exists, as shown in Figure 5-12 and Figure 5-13.

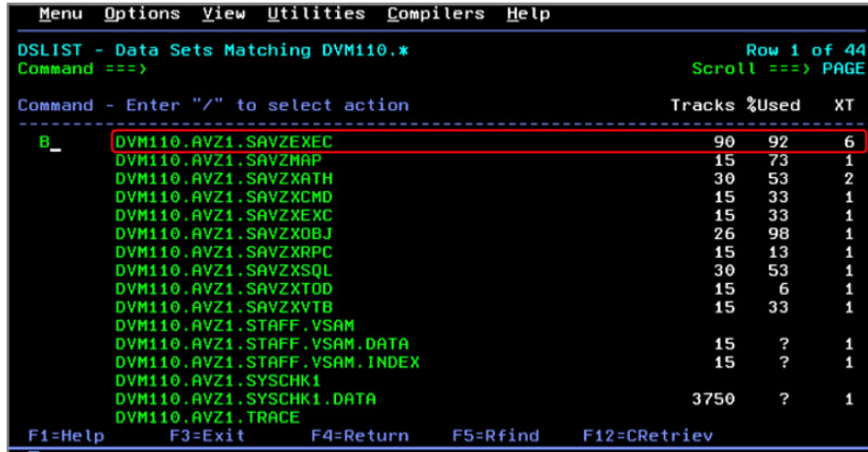


Figure 5-12 Locating DVM zVZyIN00 configuration member

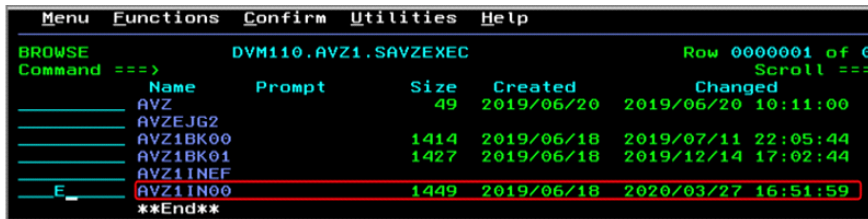


Figure 5-13 Editing DVM xVZyIN00 configuration member

The AVZ1IN00 member needs the DRDA specification that is updated for the PostgreSQL data source that is defined in the JDBC Gateway server. Adding a DEFINE DATABASE TYPE block definition after sample definitions for Db2, MSSQL, and other DRDA connection types establishes a mapping for the DVM server to connect to the PostgreSQL data source through the JDBC Gateway server. A portion of a sample DEFINE DATABASE TYPE is shown in Example 5-2.

*Example 5-2 Define Database Type*

```
"DEFINE DATABASE TYPE(JGATE)",
"NAME(PSG1)",
"LOCATION(POSTGRES)",
"DDFSTATUS(ENABLE)",
"DOMAIN(your.domain.name)",
"PORT(443)",
"IPADDR(192.168.164.74)",
"CCSID(37)"
```

This example uses the location POSTGRES and port 443 (default) as defined in the previous Admin UI configuration that was shown in 5.2.6, “Configuring data sources that use the JDBC Gateway server UI” on page 87. We selected PSG1 as our connection Name. The IPADDR parameter is set to the IP address of the system where the JGATE server was installed previously.

After the DEFINE DATABASE changes are completed, the edit session on SAVZEXEC(xVZyIN00) can be saved and closed.

## 5.2.8 Setting user credentials for the JDBC Gateway server

The DVM server provides pass-through authentication for the user that is logged in to the current session. This session uses RACF credentials by default to the remote data sources that are connected through the JDBC Gateway server. In most cases, user credentials are not the same for the remote JDBC data sources and can be mapped for each source.

Consider the following points:

- ▶ AVZDRATH is a utility that sets encrypted passwords in GLOBALU variables. This utility can be used to list credential information.
- ▶ Alternative user credentials can be set up with changes to auto-enable a SEF Rule, AVZEJGAG, and mapped with the AVZDRATH utility.
- ▶ AVZEJGAG is an ATH rule that switches credentials when accessing the JGATE data source that uses DRDA. This rule uses AES encrypted passwords that are stored as GLOBALU system variables.

## 5.2.9 Establishing secure access using AVZDRATH

DVM provides the ability to configure authorization credentials by using the SEF event facility. An authorization rule is shipped with DVM and is used to provide credentials when connecting to remote databases.

### Setting user credentials for JDBC Gateway server

The DVM server provides a pass-through authentication for the logged in user and by default establishes access that is based on that the user's credentials for the Resource Access Control Facility (RACF) on the z/OS system.

When accessing remote data sources by using the JDBC Gateway server, these RACF user credentials are referenced to access and retrieve remote data. In most cases, the user credentials across remote data sources are different. The JDBC Gateway server allows you to map user authentication and authorization individually for each targeted remote data source.

Alternative user credentials can be set up with changes to auto-enable a SEF Rule, AVZEJGAG, and mapped with the AVZDRATH utility.

To define alternative authentication information, edit the sample job for the AVZDRATH utility to add a global default user definition or authentication information for specific mainframe users.

The AVZDRATH member can be edited in the hlq.SAVZCNTL data set by adding a definition for the example PostgreSQL database to map the value from RACF to the user ID that is needed to access the PostgreSQL database. The DBTYPE=JGATE and DBNAME=*name* are required to proceed with each set of user ID mappings for a specific data source instance, as shown in Figure 5-14.

```
000136 *
000137 * Logon for POSTGRES - JGATE
000138 *
000139 DBTYPE=JGATE
000140 DBNAME=PSG1
000141 ZOSUSER=MYRACFID
000142 SETAUTH=dvmuser:dvm2019
```

Figure 5-14 Specific user authorization for PSG1 J GATE data source

After the definition entries are complete, the AVZDRATH JCL job can be submitted. AVZDRATH also provides a means for setting a DEFAULTUSER. This setting allows the Jgate Server to establish a proxy or functional ID for a larger set of RACF IDs, which provides default credentials for a JGATE-connected data source.

The example that is shown in Figure 5-15 shows a DEFAULTUSER on lines 002500 and 003800. Other keywords enable printing the SYSIN statements in the job output (ECHO=ON/OFF), and provide more detailed or summary information about the AVZDRATH utility settings in the job output (REPORT=DETAIL/SUMMARY).

```

001100 /*
001200 //SYSIN DD *
001300 *
001400 * Turn on/off printing of SYSIN statements to prevent
exposing
001500 * passwords in job output
001600 *
001700 ECHO=ON
001800 *
001900 * Report the current state of all DRDA authentication
information provided by AVZDRATH
002000 *
002100 REPORT=DETAIL
002200 *
002300 * Turn on default user
002400 *
002500 DEFAULTUSER = ENABLED
.....
.....
.....
003200 *
003300 * Logon for POSTGRES - PSG1 - JGATE
003400 *
003500 DBTYPE=JGATE
003600 DBNAME=PSG1
003700 *
003800 ZOSUSER=DEFAULTUSER
003900 SETAUTH=dvmuser:dvm2019
004000 *
004100 ZOSUSER=MYRACFID
004200 SETAUTH=dvmuser:dvm2019
004300 //

```

Figure 5-15 Report job summary for AVZDRATH settings

Providing user credential entries in the AVZDRATH utility job can introduce a security risk if unauthorized users can browse the AVZDRATH member to view user ID entries.

## 5.2.10 User access that uses rules

The DVM server provides pass-through authentication for the user that is logged in to the current session. This session uses RACF credentials by default to the remote data sources that are connected through the JDBC Gateway server. In most cases, user credentials are not the same for the remote JDBC data sources and can be mapped for each source.

Consider the following points:

- ▶ AVZDRATH is a utility that sets encrypted passwords in GLOBALU variables. This utility can be used to list credential information.
- ▶ AVZEJGAG is an ATH rule that switches credentials when accessing a JGATE data source that uses DRDA. This rule uses AES encrypted passwords that are stored as GLOBALU system variables.



## List ZOSUSER mapping in the AVZ member

Another option to listing the ZOSUSER mappings directly in the member is to concatenate a separate sequential dataset entry by using a SYSIN DD specification. The dataset that is specified in the SYSIN DD can be RACF protected to prevent unauthorized access to the user credentials, as shown in Figure 5-16 for DVM110.AVZ1.AUTH.

```
//SYSIN DD *
* Turn off printing of SYSIN statements to prevent exposing
* passwords in job output
ECHO=OFF
* Produce a summary report of actions taken
REPORT=SUMMARY
/*
// DD DISP=SHR,DSN=DVM110.AVZ1.AUTH
//

```

Figure 5-16 ZOSUSER authorization mappings

The dataset that is specified in the SYSIN DD can be RACF protected to prevent unauthorized access to the user credentials shown (as shown in Figure 5-17) for DVM110.AVZ1.AUTH.

```
DEFAULTUSER = ENABLED
*
DBTYPE=JGATE
DBNAME=PSG1
*
ZOSUSER=DEFAULTUSER
SETAUTH=dvmdef:dvm9999
*
ZOSUSER=MYRACFID
SETAUTH=dvmuser:dvm2019
```

Figure 5-17 Prevent unauthorized access that uses RACF

### 5.2.11 Using rules to ensure global user authorization

Running the AVZDRATH utility ensures that the new GLOBALU settings are referenced in the DVM server by automatically enabling the SEF ATH rule for the PDS member AVZEJGAG. This section describes how to use the TSO ISPF panel to make changes to the SEF rule for user authorization on the DVM server.

#### **Starting ISPF panels that use TSO**

In some instances, ISPF panels can be set up as a secondary menu from the ISPF Primary options menu or can be started by running the EXEC command from a TSO command line. The ISPF panel can be set up in the h1q.xVZy.SAVZEXEC library, and started by running a TSO command shell (see Example 5-18 on page 93):

```
EXEC 'h1q.xVZy.SAVZEXEC(xVZ)''SUB(xVZy)
```



```

Menu List Mode Functions Utilities Help
ISPF Command Shell
ISPF Command ==>
Enter TSO or Workstation commands below:
==> ex 'DVM110.AVZ1.SAVZEXEC(AVZ)' 'SUB(AVZ1)'

```

Figure 5-18 Starting DVM ISPF interface from TSO command line

### Updating rulesets by using the DVM server ISPF panel

By using the DVM server ISPF panel, rules can be defined to handle specific business needs. The use of Rules Management is needed for this alternative method for managing secure access to a target data source through the JDBC Gateway server, as shown in Figure 5-19.

```

IBM Data Virtualization Manager for z/OS
Option ==> E_

Interface Facilities:
1 ACI 5 IDMS SSID : AVZ1
2 Adabas 6 IMS Version : 01.01.00
3 CICS 7 VSAM/Sequential Date : 20/04/30
4 DB2 8 DSSPUFI Time : 23:20

Data Virtualization Server Administration:
A Remote User - Manage Remote Users
B Server Trace - Server Trace Facility - SIS SSID: AVZ1
C AVZ Admin. - Manage Data Virtualization Server
D Data Mapping - Data Mapping Facility
E Rules Mgmt. - Event Facility Management
F Monitor - Monitor Server Activity
G Streams - Streams Administration
H Services - Services Administration
I Instrumentation - Instrumentation Server Administration

```

Figure 5-19 Navigating to Rules Management on the DVM server

The Event Facility (SEF) provides the unique ability to customize rules for use of variables, handle data or business triggers, and monitor their use. Option 2 for SEF Rule Management steps through the creation of a rule for the ZOSUSER in SYSIN, as shown in Figure 5-20.

```

----- Event Facility (SEF) Control ----- SSID: AVZ1
Option ==> 2_

1 Global Variables - Display and Update Global Variables
2 SEF Rule Management - Control SEF Event Procedures & Libraries
 Show Selection Panel at Entry ==> Y
3 Interactive Command - Issue SEF, AVZ., or Product Rexx Commands
4 Bound Packages - Display IDF bound package information:
 Collection or * ==> *
 Package or * ==> *
 Show SQL Stmts ==> Y
 Produce Hex Dump ==> N

```

Figure 5-20 SEF Rules Management

For the SEF change, all of the default values apply for rulesets, types, directory reads, confirmations, and the entry panel. To address this rule, the ATH ruleset can be ENABLED under status. Selecting this ruleset for DVM110.AVZ1.SAVZXATH displays a list of associated PDS members.

The PDS member for the ATH rule is AVZEJGAG. Option B was selected for this member and rule to auto-enable ATH by showing a Y flag for always-on, as shown in Figure 5-21.

```
----- Event Facility (SEF) Event Procedure Lis Row 39 to 51 of 53
Command ==>
LCs: S ISPF Edit E Enable D Disable A Set Auto-Enable Scroll ==> CSR
 Z Reset Auto-Enable B Set Auto/Enable C Disable/Reset Auto

PDS Members for: DVM110.AVZ1.SAVZXATH
A
S Member Status E TYP VV.MM Created Modified Size Init Mod ID

AVZCNSR DISABLED N *** 01.01 09/07/21 17/09/29 09:28 162 118 162 AVZ
AVZEBIGG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEDASG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEDB2G DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEDDBG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEDRBG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEIFXG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEJGAG ENABLED Y ATH 01.03 17/07/14 19/07/31 18:14 185 181 24 ADMIN
AVZELUWG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 13 AVZ
AVZEHSSG DISABLED N *** 01.01 17/07/14 17/09/29 09:28 185 181 11 AVZ
```

Figure 5-21 Enabling and automatically enabling the AVZEJGAG ATH rule

After the ATH settings for the AVZEJGAG PDS member are completed and the ISPF panel exists, the DVM server started task is to be stopped and restarted.

## 5.2.12 Connecting to a JGATE Data Source in DVM Studio

In this section, we review our Data Virtualization Manager Studio to view the sample PostgreSQL connection we configured by using the JDBC Gateway server.

DVM Studio includes a Server panel with DVM server information. For this example, AVZ1 is the host DVM server that is configured and connected to the JDBC Gateway server named JGATE. Figure 5-22 shows a tree structure for the AVZ1 server with references to Virtual Tables and Virtual Views that are mapped to local data sets and remote sources that are mapped that use the JDBC Gateway server.

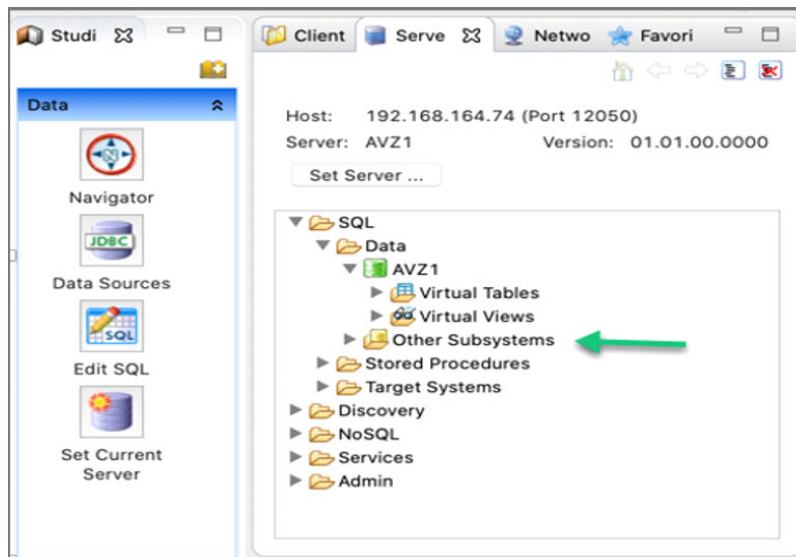


Figure 5-22 Navigating in DVM Studio to other subsystems

The tree structure also lists “Other Subsystems” that are configured for the DVM server. The current view that is shown in Figure 5-19 displays Db2 members and a series of JDBC Gateway servers that are configured across various workstations that represent Windows, Linux, and macOS. Each of the JGATE references represents database definitions that are linked to the AVZ1 DVM server, and in particular, the PSG1 (JGate) server that is used to access Postgres data remotely, as shown in Figure 5-23.

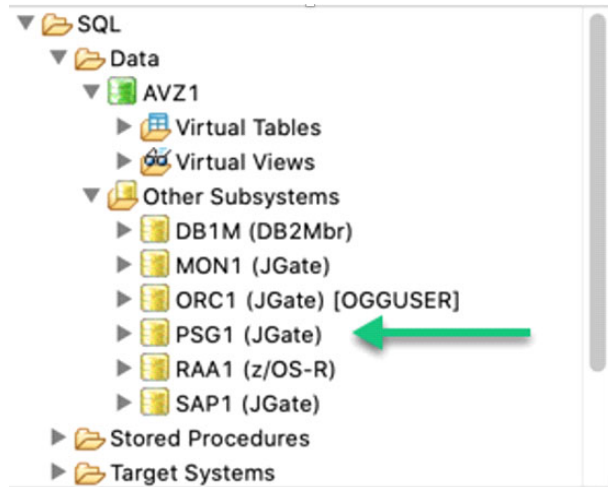


Figure 5-23 Displaying the PSG1 Postgres database running non main-frame

The DVM server maintains metadata in memory that is associated with each data source that is mapped for access. The DVM server also performs data discovery for the target data source and captures details about schema, tables, and views that are on those database systems. As a result, DVM Studio can immediately access all of the DVM server metadata and easily present database objects in a relational format for simple access by SQL API and other modern programming languages.

DVM Studio can access a remote server’s database by schema where tables and views can be virtualized. Figure 5-24 shows the emp table under the dvm\_schema. The JDBC Gateway server serves as a literal gateway for the DVM server, which then allows client applications to read/write to and from the respective data source.

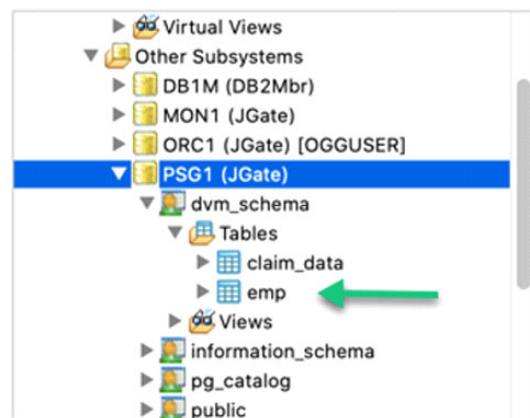


Figure 5-24 Data discovery of Postgres database

Figure 5-25 shows two tables that exist within the dvm\_schema for the PSG1 PostgreSQL database. One of the tables is named emp. DVM Studio can create a virtual table that maps to the emp table, and generates SQL and programming code (Java, Python, and so on). DVM Studio also can create and publish web services that can be used by RESTful applications.

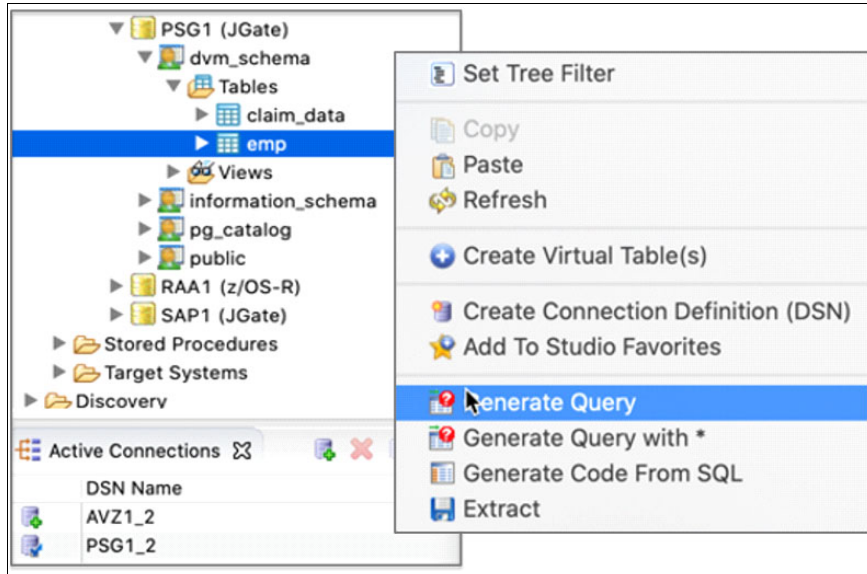


Figure 5-25 Tree structure of remote PSG1 PostgreSQL database with the emp table selected

### Generating a query

DVM Studio generates SQL statements dynamically for the selected database object, which defaults to selecting all columns and rows up to a built-in limit to the Studio, as shown in Example 5-3.

*Example 5-3 Result from SQL query in emp table in PostgreSQL*

---

```
-- Description: Retrieve the result set for emp
-- Tree Location: 192.168.164.74/12050/SQL/Data/Other Subsystems/PSG1/dvm_schema/Tables/emp
-- Remarks:
SELECT "empno", "firstname", "midinit", "lastname", "workdept", "phoneno", "job", "edlevel"
FROM "dvm_schema"."emp";
```

---

### Running and viewing generated query results

DVM Studio then can display results from \ tables, view results of a newly created virtual table, and join results from two heterogeneous database tables. Join operations can occur exclusive to relational, non-relational, or any combination of types.

The example that is in Figure 5-26 on page 97 shows the result set from the query that was run in Example 5-3. Notice that the generated query selects all columns and includes the normalized 3-part name.

|   | empno  | firstnm   | midinit | lastname  | workdept | phoneno | job | edlevel |
|---|--------|-----------|---------|-----------|----------|---------|-----|---------|
| 0 | 000010 | CHRIST... | I       | BAAS      | A00      | 3978    | P.. | 18      |
| 1 | 000020 | MICHAEL   | L       | THOMPSON  | B01      | 3476    | M.. | 18      |
| 2 | 000030 | SALLY     | A       | KHAN      | C01      | 4738    | M.. | 20      |
| 3 | 000050 | JOHN      | B       | GREYER    | E01      | 6789    | M.. | 16      |
| 4 | 000060 | IRVING    | F       | STERN     | D11      | 6423    | M.. | 16      |
| 5 | 000070 | EVA       | D       | PULASKI   | D21      | 7831    | M.. | 16      |
| 6 | 000090 | EILEEN    | W       | HENDERSON | E11      | 5498    | M.. | 16      |
| 7 | 000100 | THEODO... | Q       | SPENSER   | E21      | 972     | M.. | 14      |
| 8 | 000110 | VINCEN... | G       | IUCCHESI  | A00      | 3490    | S.. | 19      |
| 9 | 000020 | MICHAEL   | L       | THOMPSON  | B01      | 3476    | M.. | 18      |

Figure 5-26 Result from executed SQL query of emp table in PostgreSQL

### Creating a virtual table using DVM Studio

A target system must be referenced for the virtual tables from the remote JDBC source. After the target system exists for the remote database, DVM Studio discovers the database objects for the target system that can now be virtualized.

In the window that is shown in Figure 5-27, the metadata library is displayed where the mapping for the data source is stored. In this example, a 3-part naming pattern is defined for the emp PostgreSQL table that is selected.

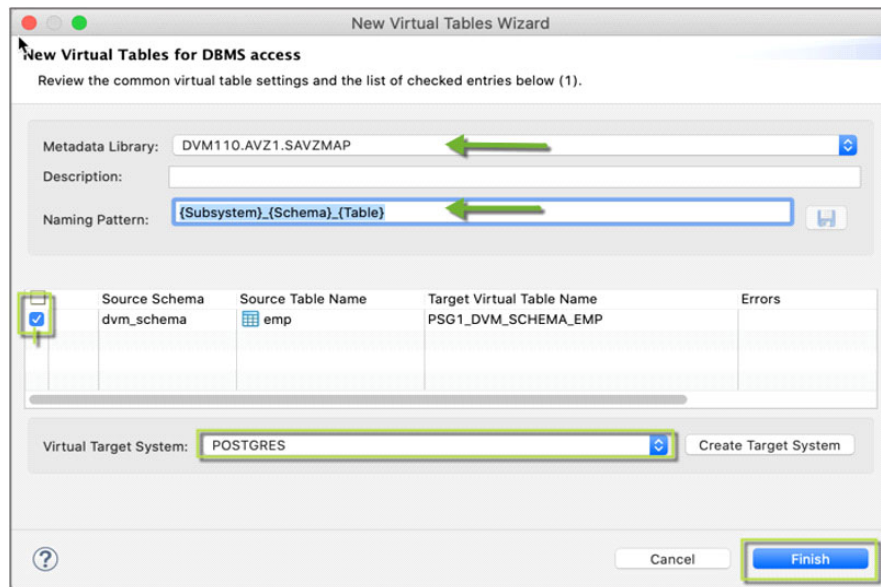


Figure 5-27 New virtual target system

After the virtual table is created, a query can be generated from the newly defined mapping, much in the same way the query was generated by accessing the data directly (see Figure 5-26).

The results from the virtual table of the emp base table display in DVM Studio output, as shown in Figure 5-28.

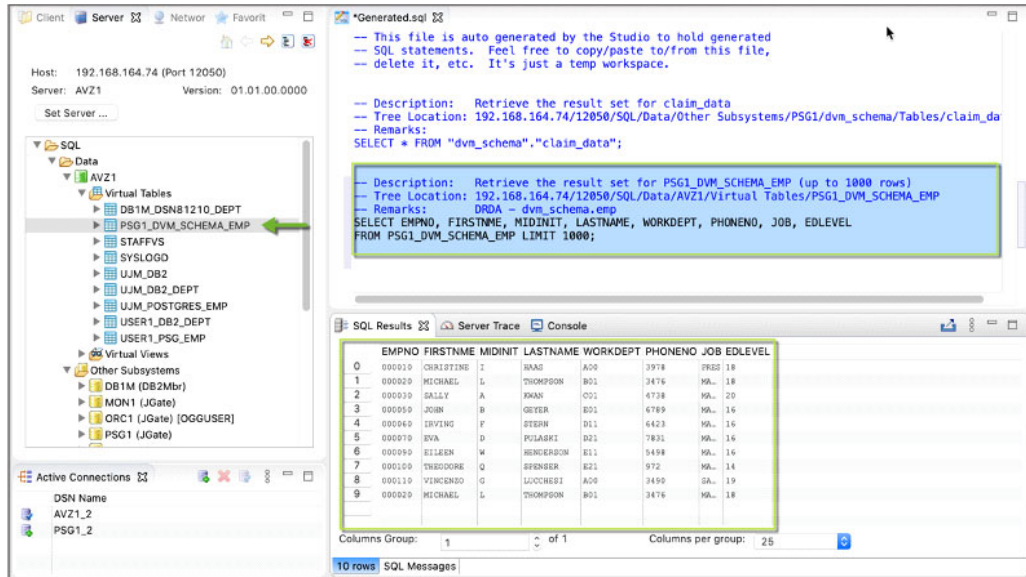


Figure 5-28 Query results from EMP virtual table

### Joining data with virtual views using DVM Studio

A virtual view that joins two tables from different data sources also can be created. In the example that is shown in Figure 5-29, a JOIN is created between DB1M\_DSN81210\_DEPT (from Db2 z/OS) and PSG1\_DVM\_SCHEMA\_EMP (from PostgreSQL).

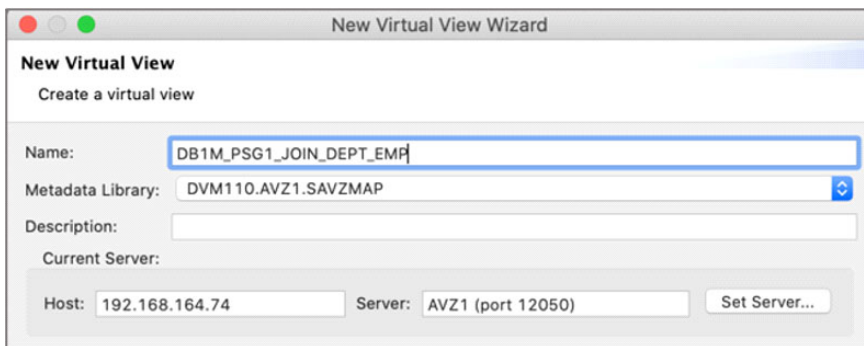


Figure 5-29 Creating a virtual view that JOINS across two Virtual Tables



With several virtual views defined, you can start with a single view and then add tables with JOIN criteria by using a free form SQL Editor, as shown in Figure 5-30.

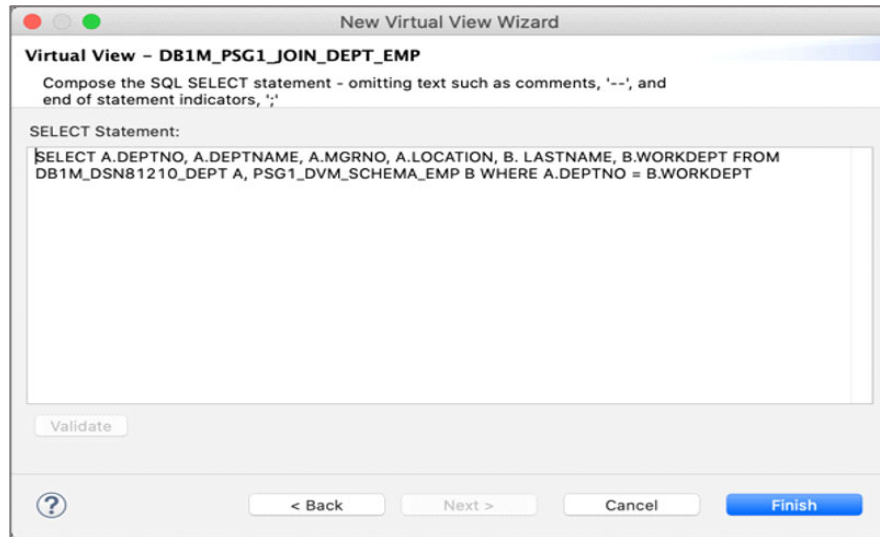


Figure 5-30 SQL SELECT JOIN between EMP and DEPT virtual tables

For this example, the DEPT table from the Db2 for the z/OS database is joined with the EMP table from the PostgreSQL database. With the Virtual View created, a query can be generated and run to view the result of the JOIN operation between the two Virtual Tables, as shown in Figure 5-31. The example shows accessing data on the mainframe and joining it with a table on a remote distributed environment through the JDBC Gateway server.

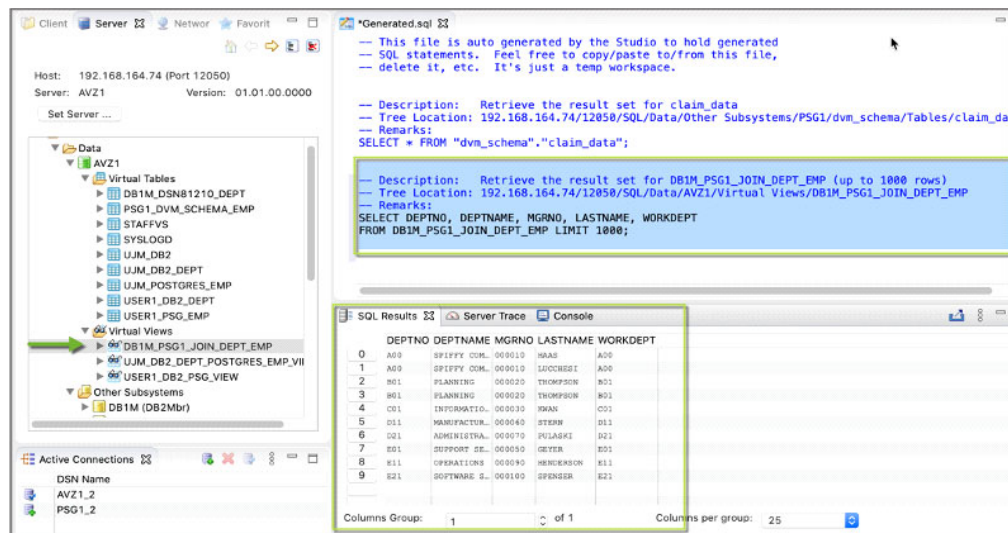


Figure 5-31 Result from SQL Query on joined tables EMP and DEPT

### 5.2.13 Secure access that uses AVZDRATH

The DVM server provides pass-through authentication for the user that is logged in to the current session that uses RACF credentials by default to the remote data sources that are connected through the JDBC Gateway server. In most cases, user credentials are not the same for the remote JDBC data sources. The DVM server can be used to encrypt a unique password for the user access to be mapped for each source.

Alternative user credentials can be set up with changes to auto-enable a SEF Rule, AVZEJGAG, and mapped with the AVZDRATH utility.

To define alternative authentication information, edit the sample job for the AVZDRATH utility to add a global default user definition or authentication information for specific mainframe users.

The AVZDRATH member can be edited in the hlq.SAVZCNTL data set by adding a definition for the example PostgreSQL database to map the value from RACF to the user ID that is needed to access the PostgreSQL database. The DBTYPE=JGATE and DBNAME=*name* are required to proceed with each set of user ID mappings for a specific data source instance, as shown in Figure 5-32.

```
000136 *
000137 * Logon for POSTGRES - JGATE
000138 *
000139 DBTYPE=JGATE
000140 DBNAME=PSG1
000141 ZOSUSER=MYRACFID
000142 SETAUTH=dvmuser:dvm2019
```

Figure 5-32 Specific user authorization for PSG1 JGATE data source

After the definition entries are completed, the AVZDRATH JCL job can be submitted. AVZDRATH also provides a means for setting a configurable DEFAULTUSER. This setting allows the Jgate Server to establish a proxy or functional ID for a larger set of RACF IDs, which provides default credentials for a JGATE-connected data source.



Figure 5-33 shows a DEFAULTUSER on lines 002500 and 003800. Other keywords allow printing of the SYSIN statements in the job output (ECHO=ON/OFF), and providing more detailed or summary information of the AVZDRATH utility settings in the job output (REPORT=DETAIL/SUMMARY).

```
001100 /*
001200 //SYSIN DD *
001300 *
001400 * Turn on/off printing of SYSIN statements to prevent
exposing
001500 * passwords in job output
001600 *
001700 ECHO=ON
001800 *
001900 * Report the current state of all DRDA authentication
information provided by AVZDRATH
002000 *
002100 REPORT=DETAIL
002200 *
002300 * Turn on default user
002400 *
002500 DEFAULTUSER = ENABLED
.....
.....
.....
003200 *
003300 * Logon for POSTGRES - PSG1 - JGATE
003400 *
003500 DBTYPE=JGATE
003600 DBNAME=PSG1
003700 *
003800 ZOSUSER=DEFAULTUSER
003900 SETAUTH=dvmuser:dvm2019
004000 *
004100 ZOSUSER=MYRACFID
004200 SETAUTH=dvmuser:dvm2019
004300 //
```

Figure 5-33 Specific user authorization for PSG1 JGATE data source

**Note:** Security risks can result if clear text user credentials are entered in the AVZDRATH member.





## Access methods

DVM for z/OS technology supports multiple APIs that can be used access to virtualized data sources. In this chapter, we discuss the different access methods that allow applications to seamlessly make requests for read or write operations of underlying data, regardless of location or format.

This chapter includes the following topics:

- ▶ 6.1, “Interface methods for client access” on page 104
- ▶ 6.2, “Standard access” on page 105
- ▶ 6.3, “DS Client” on page 109
- ▶ 6.4, “REST and SOAP Web service interfaces” on page 111
- ▶ 6.5, “Integrated Data Facility for mainframe applications” on page 122
- ▶ 6.6, “Db2 for z/OS UDTF” on page 123
- ▶ 6.7, “Db2 federation” on page 126
- ▶ 6.8, “IBM Cloud Pak for Data” on page 128

## 6.1 Interface methods for client access

DVM for z/OS technology provides multiple methods for client applications to access data. These access methods allow external applications to seamlessly make requests for READ or WRITE operations of underlying data, regardless of location or format (see Figure 6-1).

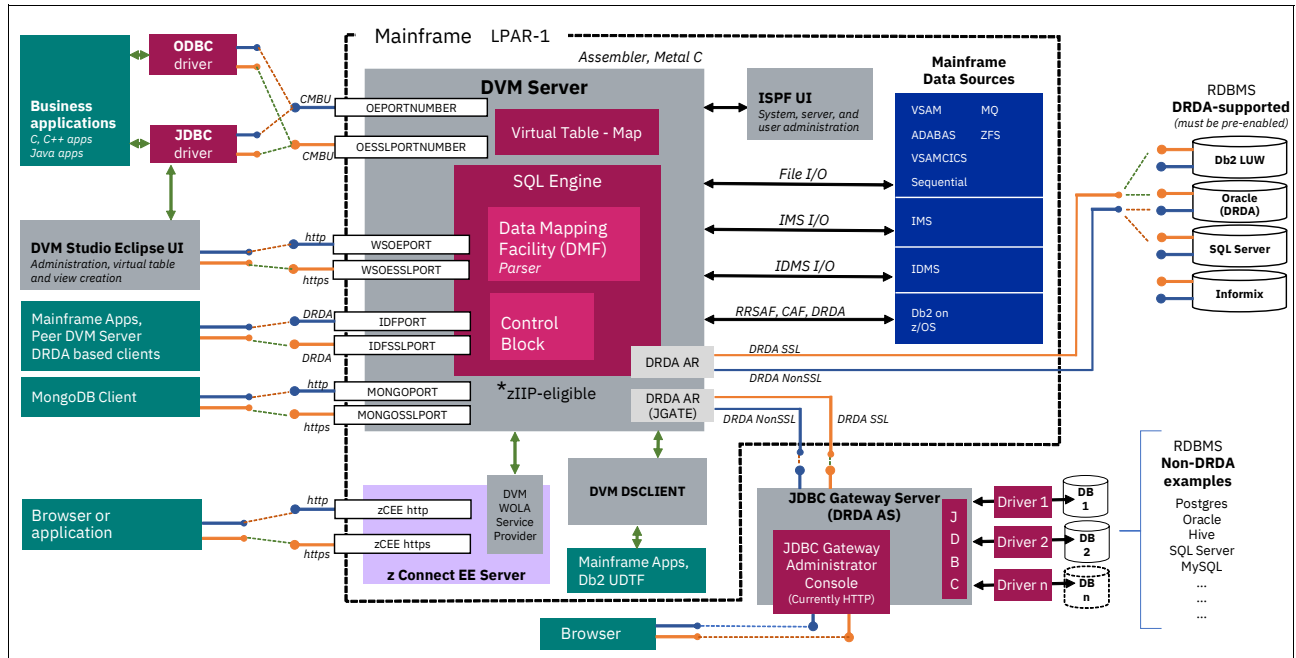


Figure 6-1 DVM for z/OS endpoint architecture

Organizations often use various applications, depending on the business objective. Traditional mainframe applications, such as Cobol, are prevalent and commonly associated with any digital transformation or modernization initiative where DVM for z/OS exists.

Modernization commonly involves transforming traditional programs into a more portable or updated interface that is driven by Java by using the DVM JDBC driver. Additionally, native and commercial applications use both DVM ODBC and JDBC drivers.

Mobile or cloud applications can interact with data that uses the web browsers and mobile applications over http or https to service cloud environments. In each instance, for any application, DVM for z/OS provides the needed translation and routing that is required for access through specific interfaces; DVM server, JDBC Gateway, DVM WOLA Service Provider, DVM DSCLIENT, Interactive System Productivity Facility (ISPF). Common client access methods are shown in Table 6-1.

Table 6-1 DVM for z/OS access methods

| Client types                                        | Access method                                                        |
|-----------------------------------------------------|----------------------------------------------------------------------|
| CC, C++ based applications                          | ODBC driver to DVM server                                            |
| Java applications                                   | JDBC driver to DVM server<br>JDBC driver to DVM Studio to DVM server |
| Mainframe applications<br>Peer DVM instance servers | IDF using DRDA                                                       |
| Mainframe applications                              | IDF using DRDA                                                       |

| Client types                                                                                           | Access method                                                                                                                 |
|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Web browser (HTTP and HTTPS)                                                                           | z/OS Connect EE to DVM WOLA service provider                                                                                  |
| Web browser (HTTTP)                                                                                    | JDBC Gateway to DRDA                                                                                                          |
| ISPF                                                                                                   | DVM server                                                                                                                    |
| RDBMS (DRDA Supported)<br>Db2 distributed family<br>Oracle (DRDA)<br>SQL Server<br>IBM Informix<br>... | DRDA to DrDA AR to DVM server                                                                                                 |
| RDBMS (non-DRDA)<br>Postgres<br>Oracle<br>Apache Hive<br>SQL Server<br>MySQL<br>...                    | JDBC driver to JDBC Gateway Server<br>JDBC Gateway Server to DVM DS Client<br>DS Client to mainframe applications or Db2 UDTF |

## 6.2 Standard access

DVM for z/OS supports ODBC and JDBC SQL access to the DVM server. The JDBC and ODBC drivers are available at IBM's [Fix Central repository](#) and can be easily downloaded. After the drivers are downloaded to a workstation, they can be decompressed and installed.

### 6.2.1 JDBC/ODBC (including security or Kerberos)

Client applications that use the JDBC driver can connect to a Db2 subsystem or other data source on or off the mainframe and access a remote system catalog or defined tables.

A minimum JDBC connection string contains the hostname, port, and DBTY, as shown in the following example:

```
jdbc:rs:dv://<hostname>:<port number>;DBTY=DVS
```

The JDBC driver `dv-jdbc-3.1.201912091012.jar` requires `log4j-api-<version number>.jar` and `log4j-core-<version number>.jar` files. It is available to use with any application by using JDBC.

An IBM Db2 QMF for Workstation contains common JDBC connection string elements, as shown in the following example:

```
jdbc:rs:dv://<hostname>:<port number>;DBTY=DVS;Subsystem=NONE
```

The DVM server can also be used as an ODBC Type 4 connection for IBM Db2 Connect. In this case, the DBTY and Subsystem parameters reference DB2, as shown in the following example:

```
jdbc:rs:dv://<hostname>:<port number>;DBTY=DB2;Subsystem=<DB2 CSSID>
```

## 6.2.2 ODBC (including security or Kerberos)

The ODBC driver is available for 32-bit and 64-bit environments across Red Hat Linux and SUSE Linux distributions. DVM Studio is the quickest approach for accessing, connecting, and discovering virtualized data asset; however, other SQL-based client programs that offer support for ODBC or JDBC connectivity are valid, such as MS-Excel, as shown in Figure 6-2.

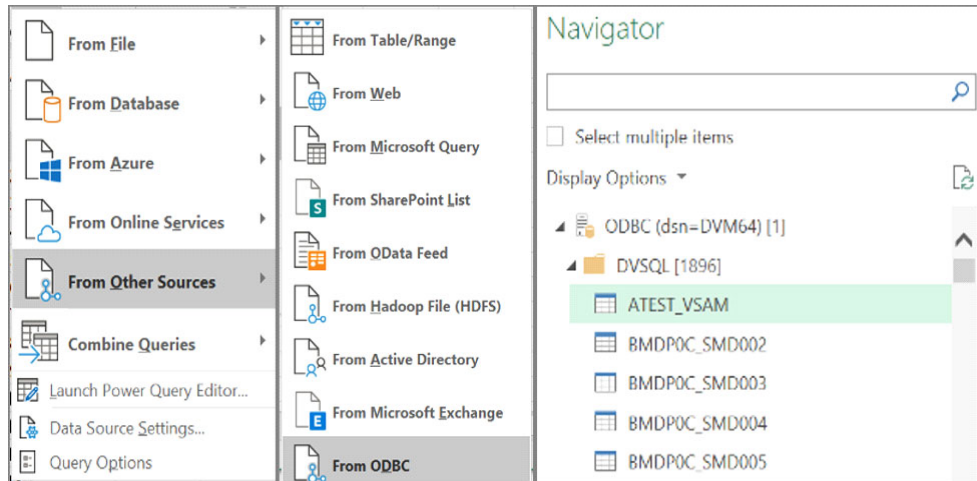


Figure 6-2 MS-Excel querying DVM

### Using DVM Studio to enhance programmer productivity

After the underlying data is formally mapped and made available by using a virtual source library, data can be easily discovered and previewed by using default connectivity to the DVM server. All virtual data assets that are provisioned through the DVM server can be used as a reference for generating customized code snippets across available modern programming languages and used for accelerating application development. Figure 6-3 shows the menu option in DVM Studio to generate programming code from SQL.

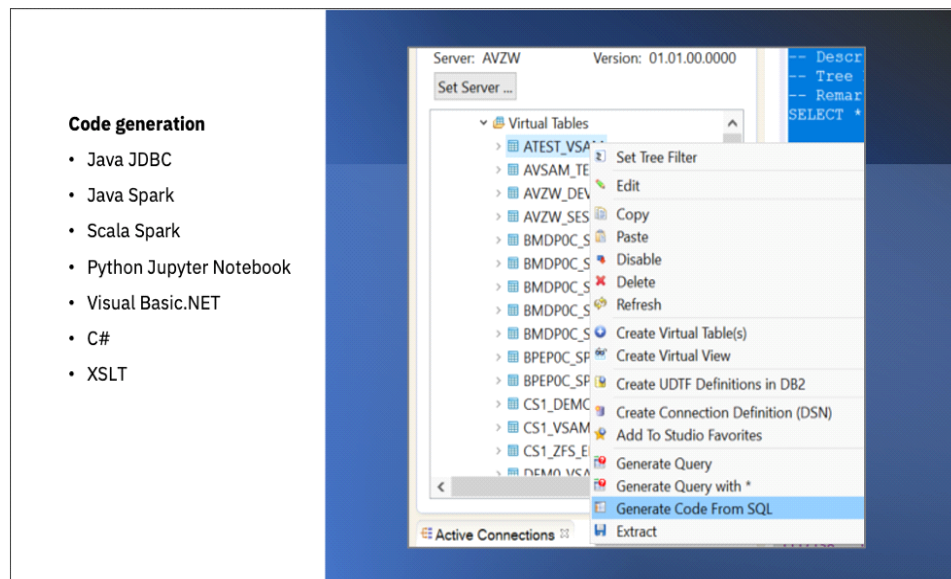


Figure 6-3 Generating sample Java code

Generated SQL code is saved in the current workspace in your Eclipse environment, as shown in Figure 6-4. By default, this code stored in the following user-profile:

C:\Users<your-id>\dvm\_workspace\Data Virtualization Manager\src.

```
import java.math.BigDecimal;
import java.sql.*;
import java.util.LinkedHashMap;

/* This class encapsulates a call to the following SQL statement.
 * <pre>
SELECT STAFFVS_KEY_IDD, STAFFVS_DATA_NAME_L, STAFFVS_DATA_NAME, STAFFVS_DATA_DEPT, STAFFVS_DATA_JOB, STAF
STAFFVS_DATA_FILLER FROM ATEST_VSAM
 * </pre>
 * <p>
 * It is written so that it may be used as a stand-alone example class,
 * or integrated as part of a larger project.
 */
public class SampleVSAM {

 // Class member variables are defined here. These represent the input
 // arguments and any result set variables that are necessary.

 // Connection given to us at initialization
 protected Connection m_connection = null;

 // Statement used to execute our call
 protected PreparedStatement m_statement = null;

 // LinkedHashMap that stores the results return return by the server.
 // Persist them in a LinkedHashMap to be able to handle server call that may returns
 // multiple result sets.
 // Sample: <result_set_1, <row_1, <colname, column_value>>>;
 protected LinkedHashMap<String, LinkedHashMap<String, LinkedHashMap<String, String>>>> m_results = null;
```

Figure 6-4 Generated Java sample code

## 6.2.3 Java application programming interface

DVM for z/OS provides a Java application programming interface (API) that includes a list of classes, interfaces with methods, fields, and constructors. These pre-written classes provide significant functions for application developers.

### Metadata

The Java API allows external tools and applications to discover DVM server objects, such as tables, views, and other object descriptions, such as column names, and column data type.

Use the standard DatabaseMetaData API to access the DVM server metadata, such as database product name, version, driver name, name of the total number of tables, and views. If you need more information about the DatabaseMetaData Interface and the methods it offers, search on the web for the Java official documentation.

Example 6-1 shows some standard configuration parameters for generating Java code snippets.

*Example 6-1 Java code snippet to retrieve all from a DVM server*

---

```
DatabaseMetaData databaseMetadata = conn.getMetaData();
String catalog = conn.getCatalog(); String schemaPattern = null;
String tableNamePattern = "%";
String[] types = null;
ResultSet tables = databaseMetadata.getTables(catalog, schemaPattern,
tableNamePattern, types);
```

---

Where:

- ▶ `tableNamePattern` is used to filter objects based on their name
- ▶ `types` can be used to filter objects based on their type, such as `TABLE` objects `String[] types = {"TABLE"};`

The `DatabaseMetaData` object is obtained by using the `getMetaData()` method of a `Connection` class. The `getTables()` method of the `DatabaseMetaData` interface is used to list all columns (see Example 6-2).

*Example 6-2 List all columns for all objects in the DVM server*

---

```
DatabaseMetaData databaseMetadata = conn.getMetaData();
String catalog = conn.getCatalog();
String schemaPattern = null;
String tableNamePattern = "%";
String colNamePattern = "%";
ResultSet tables = databaseMetadata.getColumns(catalog, schemaPattern,
tableNamePattern, colNamePattern);
```

---

Where:

- ▶ `tableNamePattern` can be used to filter objects based on their name
- ▶ `colNamePattern` can be used to filter columns based on their name

The `DatabaseMetaData` object is obtained by starting the `getMetaData()` method of the `Connection` class. The list of columns is obtained by using the `getColumns()` method of the `DatabaseMetaData` interface passing all required filters.

The DVM server metadata that uses the Java API can generate reusable code snippets for all available virtualized objects. Replace `XXX` and `YYY` to match your environment for hostname or IP and Port number. Replace `AAA` and `BBB` with valid credentials that are needed for the `getConnection()` method.

Include the following DVM for z/OS JDBC JAR file in the Java application class path:

- ▶ `dv-jdbc-[version #].jar`: DVM JDBC driver core implementation file
- ▶ `log4j-api-[version #].jar`: The logging framework API file
- ▶ `log4j-core-[version #].jar`: The logging framework implementation file
- ▶ `log4j2.xml`: A sample logging configuration file

For more information about sample reusable code, see Appendix B, “Java API sample code snippet” on page 233.



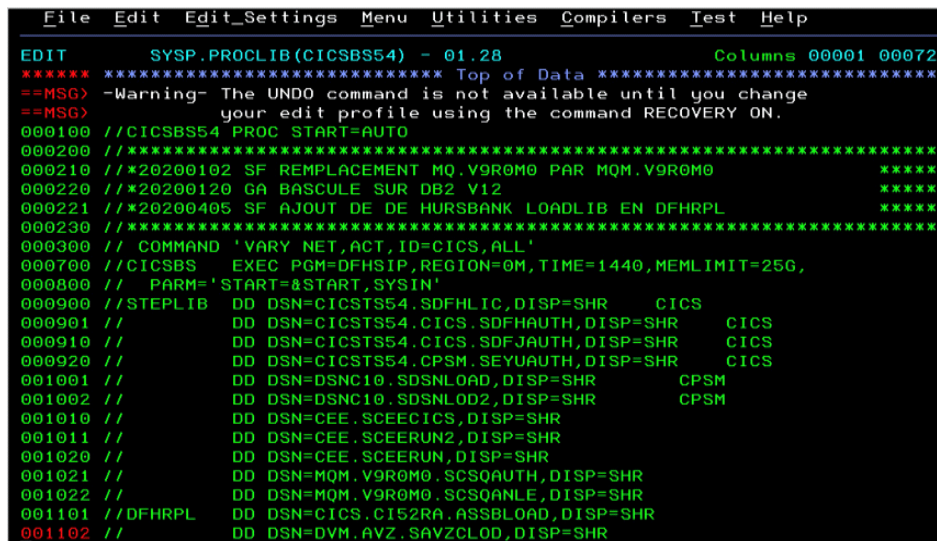
## 6.3 DS Client

The DS Client high-level API can be called from within more traditional mainframe languages, such as COBOL, Natural, or PL/I. This high-level API allows an application that is running on z/OS to use a call-level interface to communicate with the DVM server to process SQL requests and retrieve results that are buffered in a 64-bit shared memory object.

### 6.3.1 CICS and other TXN or workload balancers

To use the DsClient API with CICS, the CICS started task JCL, program list table (PLT), and DFHCSD file must be modified.

The DVM.AVZ.SAVZCLOD library must be added to the DFHRPL concatenation in each CICS region that is connecting to the DVM server for the SYSP.PROCLIB(CICBS54) data set. Figure 6-5 shows this addition by using the DVM ISPF pane.



```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT SYSP.PROCLIB(CICBS54) - 01.28 Columns 00001 00072
***** Top of Data *****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000100 //CICBS54 PROC START=AUTO
000200 //*****
000210 //*20200102 SF REPLACEMENT MQ.V9R0M0 PAR MQM.V9R0M0 *****
000220 //*20200120 GA BASCULE SUR DB2 V12 *****
000221 //*20200405 SF AJOUT DE DE HURSBANK LOADLIB EN DFHRPL *****
000230 //*****
000300 // COMMAND 'VARY NET,ACT,ID=CICS,ALL'
000700 //CICSBS EXEC PGM=DFHSIP,REGION=0M,TIME=1440,MEMLIMIT=25G,
000800 // PARM='START=&START,SYSIN'
000900 //STEPLIB DD DSN=CICSTS54.SDFHLIC,DISP=SHR CICS
000901 // DD DSN=CICSTS54.CICS.SDFHAUTH,DISP=SHR CICS
000910 // DD DSN=CICSTS54.CICS.SDFJAUTH,DISP=SHR CICS
000920 // DD DSN=CICSTS54.CPSM.SEYUAUTH,DISP=SHR CICS
001001 // DD DSN=DSNC10.SDSNLOAD,DISP=SHR CPSM
001002 // DD DSN=DSNC10.SDSNLOAD2,DISP=SHR CPSM
001010 // DD DSN=CEE.SCEECICS,DISP=SHR
001011 // DD DSN=CEE.SCEERUN2,DISP=SHR
001020 // DD DSN=CEE.SCEERUN,DISP=SHR
001021 // DD DSN=MQM.V9R0M0.SCSQAUTH,DISP=SHR
001022 // DD DSN=MQM.V9R0M0.SCSQANLE,DISP=SHR
001101 //DFHRPL DD DSN=CICS.CI52RA.ASSBLOAD,DISP=SHR
001102 // DD DSN=DVM.AVZ.SAVZCLOD,DISP=SHR
```

Figure 6-5 Add the DVM.AVZ.SABZCLOD library when using CICS

Update and assemble the CICS program list table/program initialized (PLTPI) list for the DS Client task-related user exit. The entry for the AVZXMTRI program must follow the first DFHDELIM entry in the PLTPI list to ensure that the AVZXMTRI program is run during the second stage of the CICS PLTI process.

Complete the following steps:

1. Locate the first DFHDELIM entry in the PLTP1 list:  
DFHPLT TYPE=ENTRY, PROGRAM=DFHDELIM
2. Insert the AVZXMTRI list as the second entry for the DS Client task-related user exit:  
DFHPLT TYPE=ENTRY, PROGRAM=AVZXMTRI
3. Run your CICS assembly job.

Complete the following steps to update the DFHCSD file:

1. For each CICS region, modify and submit the AVZCICSD job that is in hlq.SAVZCNTL data set.
2. Update LIST(YOURLIST) to match the start-up group list for the CICS region.

Restart CICS and check for the following message in the CICS job log:  
AVZ4459I CICSE DS Client exit program AVZCTRUE is enabledAdd body text.

### 6.3.2 Using Data Virtualization Manager in a COBOL program

Assume now that you have a VSAM data set that is virtualized as a virtual table STAFFVS, and you must access it from a program that is written in a high-level language, such as COBOL from CICS. DVM for z/OS offers a DS Client high-level language API through the program AVZCLIEN through this sequence:

1. Open a connection to the DVM server.
2. Send the SQL command to the DVM server.
3. Receive the complete query results from the issues SQL statement.
4. Close the connection to the DVM server.

#### Program's data structures

The COBOL program requires the definition of the DS Client Control Block (DVCB) fields be added to the programs Working Storage Section. These fields are used to interface with the DVM server.

For more information about the DVCB, see [IBM Documentation](#).

#### AVZCLIEN program preparation

The program is compiled without any specific options and linked as shown in the following example:

```
//SYSLIB DD DSN=DVM.SAVZLOAD,DISP=SHR
//SYSLMOD DD DSN=APPL.LOADLIB,DISP=SHR
//SYSIN DD *
INCLUDE SYSLIB(AVZCLIEN)
NAME module (R)
```

AVZCLIEN can be found in the SAVZLOAD library.

#### SQL Command Writing

Instrument the SQL command in the COBOL program by using COBOL character functions, as shown in the following example:

```
STRING
"SELECT STAFFVS_KEY_ID,
"DELIMITED BY SIZE" STAFFVS_DATA_NAME,
"DELIMITED BY SIZE" STAFFVS_DATA_DEPT,
"DELIMITED BY SIZE" STAFFVS_DATA_JOB,
"DELIMITED BY SIZE" STAFFVS_DATA_YRS,
"DELIMITED BY SIZE" FROM STAFFVS,
"DELIMITED BY SIZE" INTO SQL-TEXT.
```

The example statement results in dynamic SQL being run where STAFFVS is the virtual table requested, as shown in the following example:

```
SELECT STAFFVS_KEY_ID, STAFFVS_DATA_NAME, STAFFVS_DATA_DEPT, STAFFVS_DATA_JOB,
STAFFVS_DATA_YRS
FROM STAFFVS;
```

For more information about developing your Cobol application by using the DSClient API, see the [Developer's Guide](#).

## 6.4 REST and SOAP Web service interfaces

Cloud and mobile applications changed the way enterprises and systems interact. RESTful APIs that use JSON messages are the predominant standards for new application development. IBM z/OS Connect Enterprise Edition (zCEE) provides a framework that enables z/OS-based programs and data to participate fully in the new API economy for mobile and cloud applications.

### 6.4.1 IBM z/OS Connect Enterprise Edition

z/OS Connect EE provides RESTful API access to z/OS applications and data that is hosted in subsystems, such as CICS, IMS, IBM MQ, and Db2 for z/OS. Moreover, the combination of z/OS Connect and DVM enables direct RESTful API access to perform SELECT, INSERT, UPDATE, and DELETE operations to traditional mainframe data, such as VSAM, Sequential, SMF, Adabas, and even to non-z/OS data sources. Any data that is virtualized by using DVM for z/OS is available to z/OS Connect.

Interaction between the DVM server and z/OS Connect is enabled by the DVM Service Provider, which is a UNIX System Services component that can integrate with zCEE to enable DVM Web services invocation. In turn, the DVM Service Provider establishes the communication channel with DVM server by using WebSphere Optimized Local Adapter (WOLA) Service Provider, which is natively supplied by z/OS Connect EE.

WOLA is a function that enables fast, efficient, and low-latency cross-memory exchanges between z/OS Connect and external address spaces, such as DVM for z/OS. WOLA requires the DVM server and zCEE server to be on the same LPAR.

WOLA uses a three-part name to uniquely identify the WOLA server (or communication channel). This name is derived from the wolaGroup, wolaName2, and wolaName3 attribute values.

### 6.4.2 Configuring the DVM server for use with z/OS Connect

Allocate a WebSphere Optimizer Local Adapter (WOLA) PDSE data set with the following characteristics to configure the DVM server to work with the zCEE server. A best practice is to use the DVM server naming convention for the WOLA PDSE definition:

- ▶ Space units: TRACK
- ▶ Primary Quantity: 30
- ▶ Secondary Quantity: 2
- ▶ Directory blocks: 15
- ▶ Record format: U
- ▶ Record length: 0
- ▶ Block size: 32760
- ▶ Data set name type: LIBRARY

Start a z/OS UNIX shell and browse to the working directory:

```
cd //wlp/clients/zos
cd /usr/lpp/zosconnect/v330/wlp/clients/zos
```

Then, issue the following command to copy modules from UNIX System Services into WOLA PDSE. These modules enable WOLA communication between a DVM server and a zCEE server. You can have multiple zCEE and DVM servers that use a single WOLA PDSE data set:

```
cp -Xv ./* //'<WOLA PDSE>' "
```

Add your WOLA PDSE data set to the AVZRPCLB ddname in the server started task JCL:

```
//AVZRPCLB DD DISP=SHR,DSN=&HLQ..SAVZRPC
// DD DISP=SHR,DSN=
```

APF authorizes the <WOLA PDSE> data set and finds Enable z/OS Connect interface facility in xxxIN00 configuration member where xxx is the DVM server subsystem name:

```
/*-----*/
/* Enable z/OS Connect interface facility */
/*-----*/
if DontDoThis then do
```

Change DontDoThis in DoThis to enable the WOLA parameters and confirm that the ZCONNECT parameter is enabled.

Optionally, add the ZCONNECTPWNAMEX parameter and set the value to the concatenation of WolaName2 and WolaName3, separated by a dot (<WolaName2>.<WolaName3>). Default values for WolaName2 and WolaName3 are NAME2 and NAME3. WolaName2 and WolaName3 can be arbitrary 1 - 8 characters strings, as shown in the following example:

```
"MODIFY PARM NAME(ZCONNECTPWNAMEX) VALUE(NAME2.NAME3) "
```

**Tip:** WolaName2 and WolaName3 also are used during zCEE WOLA configuration. Therefore, if specified, it is recommended to make note of them.

Customize the **DEFINE ZCPATH** command, which is used to define a connection to a specific z/OS Connect region (server):

```
"DEFINE ZCPATH",
" NAME(ZCNALL) ",
" RNAME(DVJR1) ",
" WNAME(DVJG1) "
```

NAME is an arbitrary name and RNAME is up to 12 characters. The WNAME is up to 8 characters for the WolaGroup name.

**Tip:** RNAME and WNAME are also used during zCEE WOLA configuration; therefore, it is recommended to make note of them.

By default, DVM for z/OS retries failed connections with the zCEE server. Sometimes, failed connections are caused by an inactive zCEE server, but the communication error is always logged in the DVM server traces file. If you use RACF, a profile definition of CBIND resource class is required to allow the zCEE server and the DVM server to connect and function properly.

Consider the following points:

- ▶ A generic or discrete CBIND resource definition is required. A generic definition uses a CBIND class profile of BBG.WOLA.<WolaGroup>.\* with UACC(READ), whereas a discrete definition uses a CBIND class profile of BBG.WOLA.<WolaGroup>.<WolaName2>.<WolaName3> with UACC(READ).

- ▶ The WolaGroup is the WNAME that is specified in the **DEFINED ZCPATH** command, WolaName2 and WolaName3 are the values that are defined in the **ZCONNECTPWNAMEX** parameter. If **ZCONNECTPWNAMEX** is not defined, the WolaName2 value must be NAME2 and the WolaName3 value must be NAME3.

### 6.4.3 Installing the DVM Service Provider

After the zCEE server is defined with security authentication enabled, complete the following steps to install the DVM Service Provider that is included in hlq.SAVZBIN(AVZBIN4):

1. Transfer the member hlq.SAVZBIN(AVZBIN4) to your workstation in binary mode.
2. Rename the file to com.rs.dv.zosconnect.provider.feature\_1.0.0.esa.
3. Copy the file to a UNIX System Services directory.
4. Change the directory to //wlp/bin, where // is the path directory for your z/OS Connect EE installation. For example, change to the following directory:

```
cd /usr/lpp/zosconnect/v330/wlp/bin
```

5. Set the JAVA\_HOME = environment variable to the path of your 64-bit IBM Java SDK:  
export JAVA\_HOME=/usr/lpp/java/IBM/J8.0\_64
6. Set the WLP\_USER\_DIR environment variable to the location where your server instances and user features are stored:

```
export WLP_USER_DIR=/var/zosconnect
./installUtility install
/dvmServiceProvider/com.rs.dv.zosconnect.provider.feature_1.0.0.esa
```

7. The zCEE server.xml can be edited by using IBM Explorer for z/OS.

Under xml element, add the following lines:

```
<feature>usr:dvsProvider</features>
<feature>zosLocalAdapters-1.0</feature>
```

Where usr:dvsProvider is DVM Service Provider on zCEE. zosLocalAdapters-1;0 is the WOLA provider that is used to communicate between the DVM Service Provider on zCEE and DVM server.

8. Under the <server> xml element, add entries for the required DVM Service Provider on zCEE:

```
<zosconnect_zosConnectService
 id="zosConnectDvsService"
 serviceName="DvsService"
 serviceRef="dvsService"
 serviceDescription="IBM DV Service provider"
 invokeURI="/dvs" />

 <usr_dvsService
 id="dvsService"
 connectionFactoryRef="wolaCF"
 registerName="DVJR1"
 serviceName="DVJS1"
 invokeURI="/dvs" />
```

Where:

- serviceRef in <zosconnect\_zosConnectService> must be the same as ID in <usr\_dvsService>.

- invokeURI in <zoscconnect\_zosConnectService> must be the same as invokeURI in <usr\_dvsService>. It represents the “root” directory of DVM Web services, which means that all DVM Web services are reachable with a URL and starts with http(s)://<zos\_connect\_ip>:<zos\_connect\_port>/dvs/.
  - connectionFactoryRef in <usr\_dvsService> must have the same value as connectionFactoryRef in <zoscconnect\_localAdaptersConnectService> and ID in <connectionFactory>, which are defined in the next step.
  - registerName in <usr\_dvsService> is the same as RNAME that is defined in the **DEFINE ZCPATH** command in the DVM IN00 configuration member.
  - serviceName in <usr\_dvsService> must have the same value as serviceName in <zoscconnect\_localAdaptersConnectService>, which is defined in the next step.
9. Under <server> xml element, add the entries that are shown in Example 6-3 that are required for the WOLA service providers.

*Example 6-3 Required entries for WOLA service providers*

---

```

<zosLocalAdapters
 wolaGroup="DVJG1"
 wolaName2="NAME2"
 wolaName3="NAME3" />

 <connectionFactory id="wolaCF" jndiName="eis/ola">
 <properties.ola />
 </connectionFactory>

 <zoscconnect_zosConnectService
 id="sdef1"
 serviceName="dvs1"
 serviceAsyncRequestTimeout="600s"
 serviceRef="svc1" />

<zosLocalAdapters
 wolaGroup="DVJG1"
 wolaName2="NAME2"
 wolaName3="NAME3" />

 <connectionFactory id="wolaCF" jndiName="eis/ola">
 <properties.ola />
 </connectionFactory>

 <zoscconnect_zosConnectService
 id="sdef1"
 serviceName="dvs1"
 serviceAsyncRequestTimeout="600s"
 serviceRef="svc1" />

<zoscconnect_localAdaptersConnectService
 id="svc1"
 registerName="DVJRI"
 serviceName="DVJS1"
 connectionFactoryRef="wolaCF"
 connectionWaitTimeout="7200" />

```

---

Where:

- wolaGroup in <zosLocalAdapters> includes the same value as WNAME defined in: DEFINE ZCPATH command in the DVM IN00 configuration member
- wolaName2 and wolaName3 have the same value as WNAME defined in the ZCONNECTPWNAMEX parameter in the DVM IN00 configuration member. If ZCONNECTPWNAMEX is not defined in DVM IN00 configuration member, WolaName2 value must be NAME2 and WolaName3 value must be NAME3.
- id in <connectionFactory> is the same as connectionFactoryRef in <zosconnect\_localAdaptersConnectService> and <usr\_dvsService>.
- serviceRef in <zosconnect\_zosConnectService> must have the same value as id in <zosconnect\_localAdaptersConnectService>.
- registerName in <zosconnect\_localAdaptersConnectService> is the same as RNAME that is defined in the **DEFINE ZCPATH** command in the DVM IN00 configuration member.
- serviceName in <zosconnect\_localAdaptersConnectService> must have the same value as serviceName in <usr\_dvsService>.

#### 6.4.4 Creating zCEE RESTful APIs for access to the DVM server

When a RESTful API requests mainframe data, z/OS Connect communicates the request to the DVM server by using the WebSphere Optimized Local Adapter (WOLA). The DVM server runs the requested web service to get data.

After the configuration steps are completed, recycle the DVM and z/OS Connect Started tasks to establish a connection between the two servers. A successful pairing is written to the DVM server log:

```
19.10.58 STC05152 AVZ4502H ZCPRHUPR subtask is active
19.10.58 STC05152 AVZ4502H ZCPRHLWR subtask is active
```

#### Setting REST z/OS Connect web services preferences

REST z/OS Connect web services preferences are required when starting and running a web service request for z/OS Connect by using DVM Studio. Set web services preferences to REST using z/OS Connect and provide a Service Provider URL by using the following nomenclature:

```
https://<zos_connect_ip>:<zos_connect_port>/<invokeURI>
```

Where <invokeURI> is as represented in the server.xml.

Figure 6-6 shows the settings for a RESTful configuration.

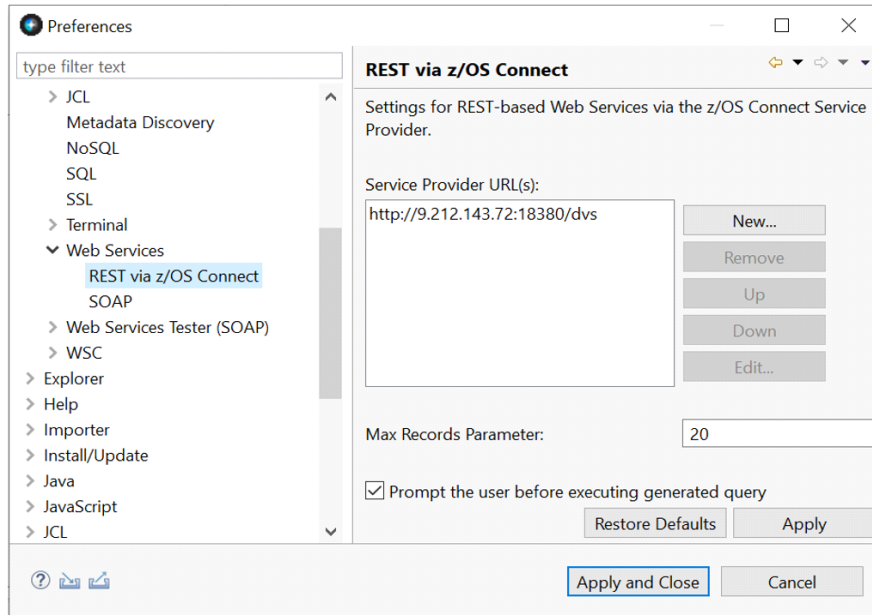


Figure 6-6 Preferences setting for REST-based web services with z/OS Connect Service Provider

Use the Max Records Parameter to set a limit for the number of records that is retrieved when run and use the Prompt user before running the generated query to prompt users before query execution.

### Assigning DVM servers where web services are run

Use the Target Systems Wizard in DVM Studio to identify a DVM server where the service provider starts web service requests. More than one target system can be defined to suit your specific configuration, as shown in Figure 6-7.

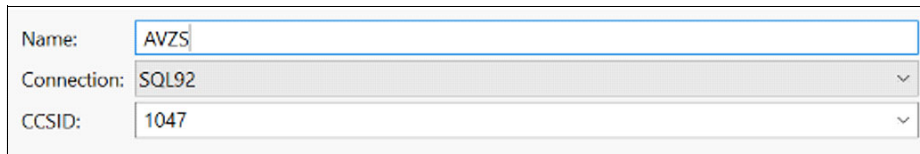


Figure 6-7 Target System creation

### Creating the web services metadata repository

Use the Web Services Directory Wizard to define Partitioned Data Sets (PDS) on the mainframe where web services metadata is stored. You provide a name for your web services directory, which is a high-level qualifier to use as a data set prefix as you create, verify, or edit metadata that is associated with web services for your mainframe system.



The library for the web services metadata is automatically created if it does not exist. Figure 6-8 shows the Web services directory definition.

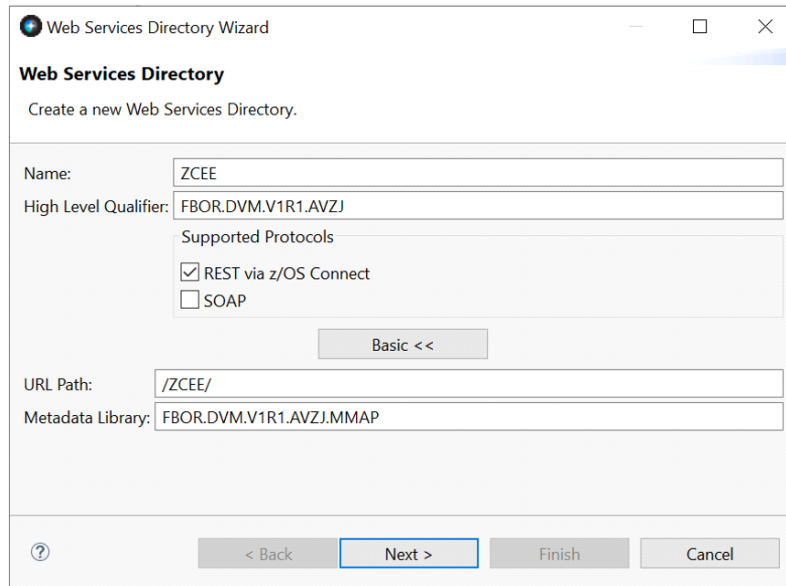


Figure 6-8 Web services directory definition

Use the Microflow Library Dataset dialog to create a microflow metadata library on your mainframe, as shown in Figure 6-9.

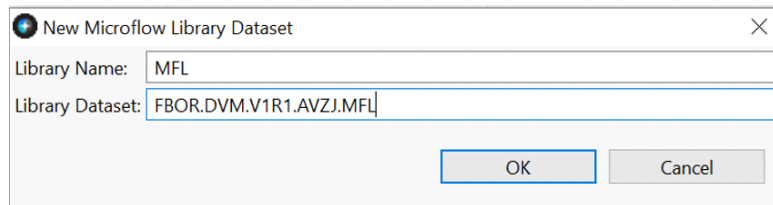


Figure 6-9 Microflow library dataset generation

### Creating web services by using DVM Studio

Use the Web Services Wizard to create a web service for your specific DVM server's web directory. The newly created web service metadata is written to your DVM server web directory. This wizard steps through assigning a name, web service operation type (REST using z/OS Connect, and business and window-level SOAP option).

You can select a virtual table or stored procedure to associate with the web service you are creating and then have the option to update the name, description, or SQL statement that is used to put or get data from a web-based client request.

Figure 6-10 shows the SQL editor element for defining a web service.

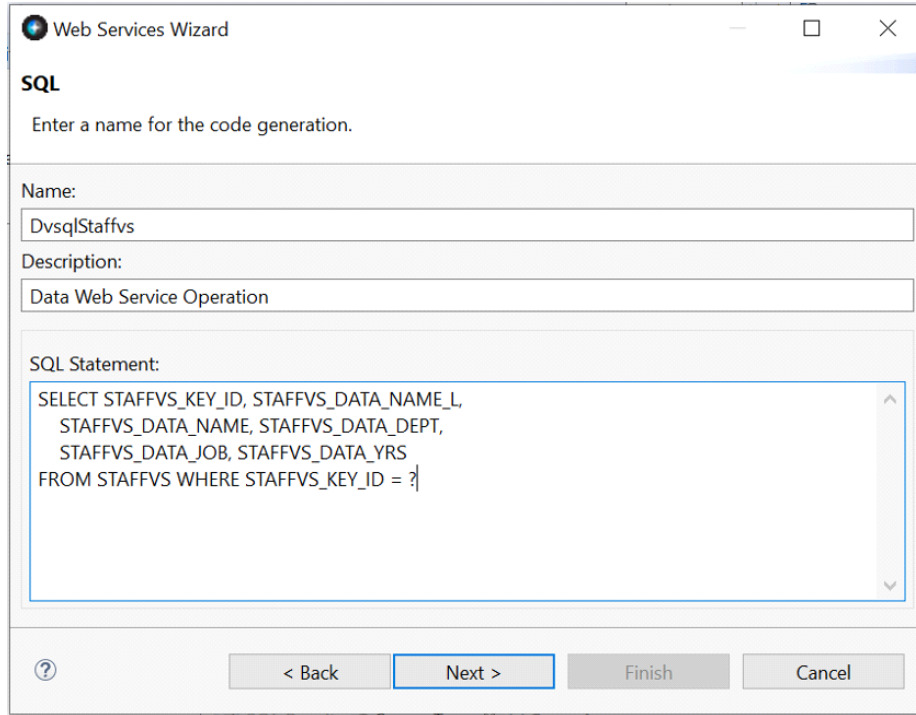


Figure 6-10 Web service definition: SQL editor

The web services workflow accommodates dynamic inputs for the SQL statement for your web service with the z/OS Connect REST interface by refreshing communication between the DVM server and the zCEE Server, as shown in Figure 6-11 and Figure 6-12 on page 119.



Figure 6-11 Web services sample list output

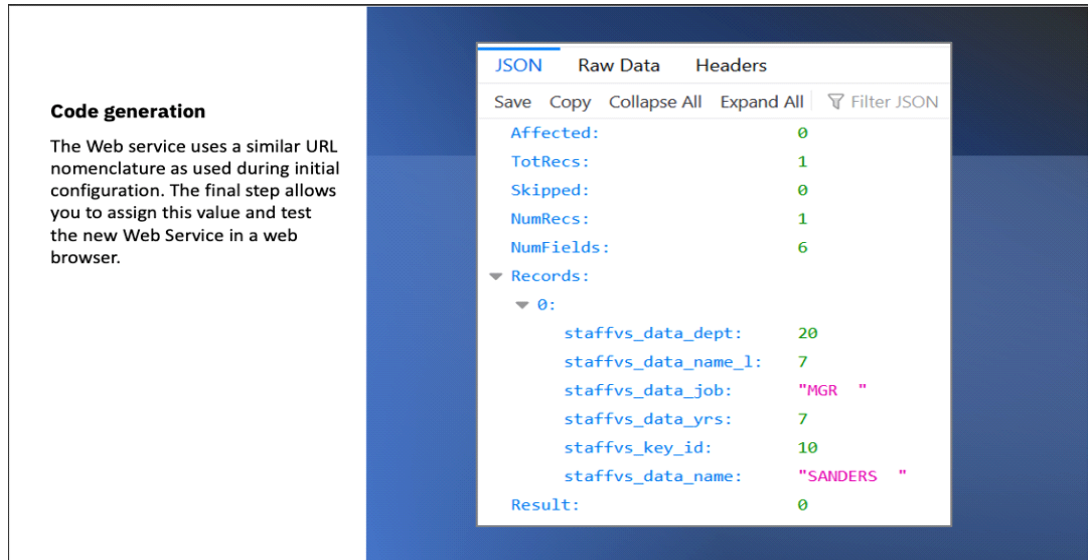


Figure 6-12 Web services sample list output

## Deploying DVM web services to z/OS Connect RESTful API

To define and deploy a z/OS Connect RESTful API, DVM Studio and an Eclipse IDE with a z/OS Connect EE API Toolkit are used to build and make available SAR files. The IBM Explorer for z/OS can use Eclipse-based plug-ins from DVM Studio and API Toolkit from zCEE to product modern applications.

You can promote a web service by generating SAR files. In DVM Studio, web services can be selected and transformed into a z/OS Connect RESTful API on the Server tab by right-clicking the wanted web service and selecting **Create z/OS Connect SAR file**.

The hypothetical DvsqIStaffvs web service uses the zCEE SAR file Generator to convert from stand-alone use to general consumption by using the zCEE REST Interface. Figure 6-13 shows SAR file generation.

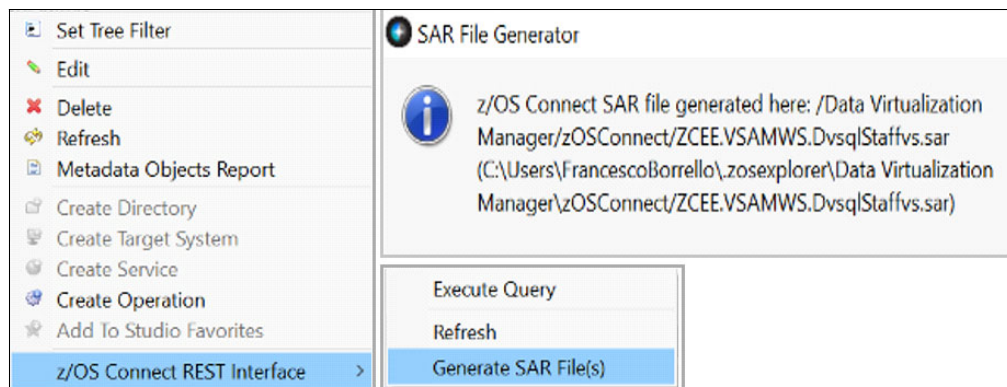


Figure 6-13 SAR File generation

By using the z/OS Connect EE API Toolkit, create a project from the Project Explorer dialog. The Editor API dialog defines the name of the service, its path, and version, as shown in Figure 6-14.

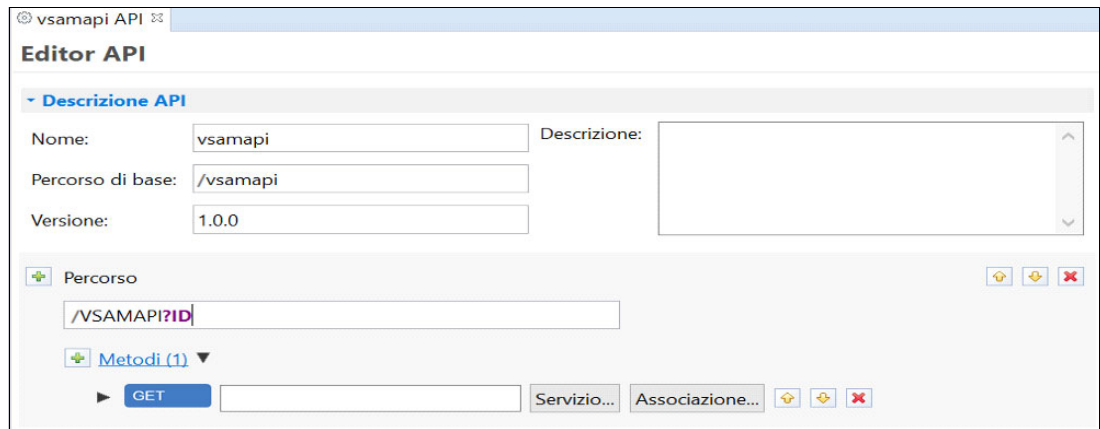


Figure 6-14 REST API path and input parameters

For each method, define the associated SAR file by using the **Service** button and locating the SAR file that was generated in your file system. Figure 6-15 shows how to associate a file to a RESTful API.

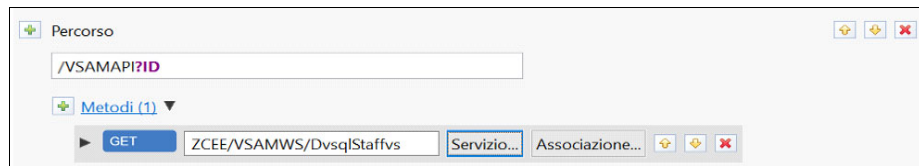


Figure 6-15 SAR file association to a REST API

### Using dynamic user input parameters

If any user input is available for the web service, use Request Mapping to define and link any input parameters that correspond to the web service parameters that are defined on DVM Studio. Finally, deploy your RESTful API by clicking the icon that is indicated in Figure 6-16.

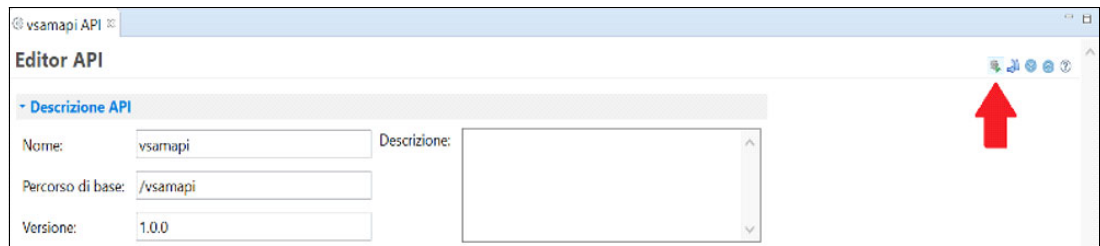


Figure 6-16 RESTful API deployment

## 6.4.5 Db2 Query Management Facility API

DVM for z/OS features a unique synergy with ZCEE in that RESTful web services can be created to access and submit requests against virtual tables that map to underlying unstructured data that is on the mainframe.

Db2 Query Management Facility (QMF) provides a RESTful API that can use DVM for z/OS by using the distributed DVM JDBC driver. Sample output from a JavaScript application is shown in Example 6-4. The output was created from Db2 SMF that calls the QMF REST API to run a query against SMF records.

*Example 6-4 SQL sub-select used to access an SMF record*

```

SELECT SMF30JBN AS JOB_NAME, SMF30STM AS STEP_NAME, SMF30PGM AS PROGRAM, SMF_SSI
AS TYPE, SMF30CSU AS CPU
FROM (SELECT SMF_STY, SMF30HPT, SMF30IIP, SMF30RCT, SMF30HPT, SMF30JQT,
SMF30RQT,SMF30HQT, SMF30SQT, SMF30STI, SMF_TIME,
SMF30SIT, SMF30IO, SMF30PSN,SMF30MSO, SMF30SRB, SMF30CSU, SMF30SRV, SMF30ISB,
SMF30ICU, SMF30PGM,SMF30CL8, SMF30SSN,
SMF30JBN, SMF30JNM, SMF30STM, SMF30STN, SMF_SSI,SMF_SID, SMF30CPT, SMF30CPS,
SMF30CPT, SMF30CPS,SMF30_TIME_ON_IFA,
SMF30_TIME_ON_SUP, SMF30TCN, SMF30TEP, SMF30SCC, LS_TIMESTAMP_LOCAL FROM
SMF_03000_SMF30IDA INNER JOIN
SMF_03000_SMF30CAS B ON A.BASE_KEY = B.BASE_KEY INNER JOIN SMF_03000 C ON
A.BASE_KEY = C.BASE_KEY INNER JOIN
SMF_03000_SMF30CMP D ON A.BASE_KEY = D.BASE_KEY INNER JOIN SMF_03000_SMF30URA E ON
A.BASE_KEY = E.BASE_KEY INNER JOIN SMF_03000_SMF30PRF F ON A.BASE_KEY = F.BASE_KEY
) WHERE SMF_STY = 4;

```

Figure 6-17 shows results from the defined web service that was generated from the Db2 QMF RESTful API that are now available to help drive operational analytics and opportunities for optimization for the application and use of resources.

|    | A               | B                | C              | D           | E          |
|----|-----------------|------------------|----------------|-------------|------------|
| 1  | <b>JOB_NAME</b> | <b>STEP_NAME</b> | <b>PROGRAM</b> | <b>TYPE</b> | <b>CPU</b> |
| 2  | SSHD3           | STEP1            | BPXPRFC        | OMVS        | 990        |
| 3  | TS59414         | STEP1            | BPXPRFC        | OMVS        | 105        |
| 4  | TS5941          | STEP1            | BPXPRECP       | OMVS        | 149        |
| 5  | IZPSRV7         | STEP1            | BPXPRFC        | OMVS        | 176        |
| 6  | DOESRV67        | STEP1            | BPXPRFC        | OMVS        | 123        |
| 7  | IZPSRV8         | STEP1            | BPXPRFC        | OMVS        | 186        |
| 8  | DOESRV68        | STEP1            | BPXPRFC        | OMVS        | 102        |
| 9  | IZPSRV9         | STEP1            | BPXPRFC        | OMVS        | 198        |
| 10 | DOESRV69        | STEP1            | BPXPRFC        | OMVS        | 110        |
| 11 | TS59416         | STEP1            | BPXPRFC        | OMVS        | 1185       |

*Figure 6-17 Results of SMF query in a spreadsheet*

## 6.5 Integrated Data Facility for mainframe applications

DVM for z/OS introduced the Integrated Data Facility (IDF), which supports a DRDA server interface. This support enables Db2 for z/OS to read or JOIN any virtual data sources that are provisioned on the DVM server that has a registration in SYSIBM.LOCATIONS and SYSIBM.IPNAMES.

### 6.5.1 DVM server subsystem in the Db2 communications database

In this case, the DVM subsystem is AVZW. The DVM server has a valid virtual table that is named SASAPPLICANT with a schema having a default three-part name of RS01AVZW.DVSQL.SASAPPLICANT.

Figure 6-18 shows a select statement that uses a three-part name of schema.database.table:

```
SELECT * FROM RS01AVZW.DVSQL.SASAPPLICANT
```



| TEMPID | NAME      | ADDRESS          | EDLEVEL | COMMENTS             |
|--------|-----------|------------------|---------|----------------------|
| 400    | FROMMHERZ | SAN JOSE, CA     | 12      | NO SALES EXPERIENCE  |
| 410    | JACOBS    | POUGHKEEPSIE, NY | 16      | GOOD CANDIDATE FOR V |
| 420    | MONTEZ    | DALLAS, TX       | 13      | OFFER SALES POSITION |
| 430    | RICHOWSKI | TUCSON, AZ       | 14      | CAN'T START WORK UN  |
| 440    | REID      | ENDICOTT, NY     | 14      | 1 YEAR SALES EXPERI  |
| 450    | JEFFREYS  | PHILADELPHIA, PA | 12      | GOOD CLERICAL BACKG  |
| 460    | STANLEY   | CHICAGO, IL      | 11      | WANTS PART-TIME JO   |

Figure 6-18 Selecting DVM data with three-part names using DB2

The three-part name can be simplified by creating a View or an Alias. However, as shown in Figure 6-18, Db2 QMF for TSO is used to generate output. The use of IDF makes DVM server virtual tables available by using three-part names or by way of a View or Alias to other mainframe applications, such as Cobol, Db2 Stored Procedures, Db2 for z/OS Restful Services, and SPUFI.

### 6.5.2 Use cases

Mainframe applications are the primary use case with limited testing against non-Z sources. Typical non-Z applications, such as MS-Excel, QMF for Workstation, IBM Cognos®, IBM SPSS®, and other client applications often use the DVM JDBC or ODBC driver through the DVM server directly.

#### Db2 as an entry point

Using Db2 as an entry point allows you to take advantage of skills and use applications that are in place as templates for new applications. Consider the following use cases:

► Use Case 1: TSO or SPUFI

QMF for TSO or SPUFI can query the DVM data with three-part names.

► Use Case 2: Cobol applications

Custom Cobol programs can be used to pull data by using embedded SQL with three-part names (or View or Alias). However, when the packages for the SQL are bound, they also must be bound in the DVM server instance that is associated with IDF. The same situation is faced with three-part names accessing another remote Db2 for the z/OS subsystem.

The job that compiles the program includes a step to \ perform a remote bind or copy the packages to the instance of DVM that is associated with IDF.

► Use Case 3: Db2 Stored Procedures

Db2 Stored Procedures can be created that run queries that are aimed at DVM data sources by using IDF. SQL Call statements to these Db2 Stored Procedures can then be incorporated into DB2 Restful Services.

### 6.5.3 Choosing Db2 UDTF or IDF

DVM's UDTFs and IDF appear similar. Therefore, how do you decide which to use? Table 6-2 shows a comparison between the two access interfaces.

Table 6-2 Db2 UDTF versus IDF

| Db2 UDTF                                                                                                                             | IDF                                                                                                                                                                                                               |
|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Can be created by any valid user with access to DVM Studio and Db2 for z/OS. No setup required.                                      | IDF must be configured by the administrators for the DVM server and Dvb2 for z/OS by using the Db2 communications database. The DVM server must be registered to populate metadata with suitable user privileges. |
| SELECT                                                                                                                               | SELECT, SELECT INTO, INSERT, UPDATE, DELETE                                                                                                                                                                       |
| By default, passes only the SELECT portion of the query to the DVM server. No SQL pushdown of WHERE predicates                       | IDF passes the SQL through the DVM server with ANSI-SQL 92 support. Also supports WHERE predicates and an SQL pushdown.                                                                                           |
| Supported by all Db2 for z/OS clients on the mainframe and over the distributed environment. For example, QMF, Cognos, and MS-Excel. | IDF is limited to mainframe client applications.                                                                                                                                                                  |

## 6.6 Db2 for z/OS UDTF

Db2 for z/OS includes a database function that is called a *user-defined table function* (UDTF). A UDTF depends on a custom program that is started by Db2 when a UDTF is called as part of a SELECT statement. This custom program can access Db2 data or it can directly access the underlying file system that is backing the Db2 database. Then, it returns the results in the normalized relational format as though seamlessly part of the local Db2 database.

You can use the Db2 for z/OS database as an information or federation hub for your enterprise. DVM Studio allows you to create Db2 UDTFs for a virtual table. These external Db2 UDTF functions provide read only support and are created within the studio on virtual tables and virtual views. These functions expose your applications to data sources that are outside of your Db2 subsystem, which allows Db2 to become a central data store for disparate data that is on or off the mainframe, such as relational, big data, Kafka, or flat files.

Db2 UDTFs that point to DVM virtual tables can be created by using DVM Studio (see Figure 6-19) and referenced as a UDTF or View in the local Db2 database.

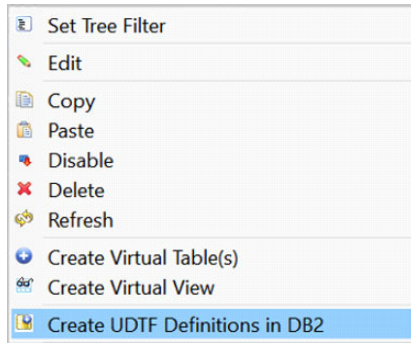


Figure 6-19 Creating a UDTF

As of this writing, the following optimized UDTF modules are available for different hardware environments:

- ▶ AVZUDT9N for the z196 system
- ▶ AVZUDTAN for the zEC12 system
- ▶ AVZUDTBN for the IBM z13® system
- ▶ AVZUDTCN for the z14 system
- ▶ AVZUDTDN for the IBM z15™ system

Db2 for z/OS Subsystem uses the Db2 Workload Manager environment for User Defined Functions (UDF) and User Defined Routines (UDR). The Db2 Wizard can be used to generate UDTF definitions or Views for the Db2 for the z/OS database that is referencing IBM Z provisioned data on the DVM server, as shown in Figure 6-20.

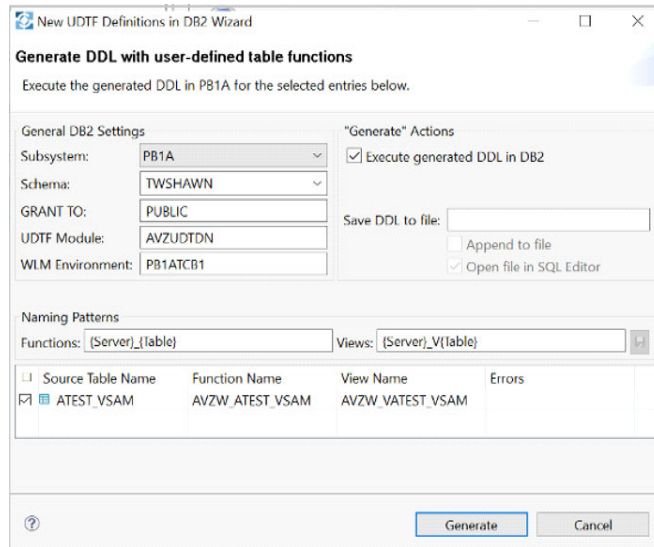


Figure 6-20 Creating a UDTF using the Db2 Wizard



The SQL results profile the Db2 UDTF execution trace. The UDTF is now present in SYSIBM.SYSROUTINES on the designated Db2 Subsystem, as shown in Figure 6-21.

```

10:09:34:857, 21 Apr 2020 - Start SQL Execution.
10:09:34:857, 21 Apr 2020 - CREATE FUNCTION "TWSHAWN"."AVZW_SASAPPLICANT" (CONDITION VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, DVMNAME VARCHAR(254) CCSID EBCDIC FOR SBCS DATA, COLINFO VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, REQUEST VARCHAR(254) CCSID EBCDIC FOR SBCS DATA) RETURNS TABLE ("TEMPID" SMALLINT, "NAME" VARCHAR(9), "ADDRESS" VARCHAR(17), "EDLEVEL" SMALLINT, "COMMENTS" VARCHAR(29)) EXTERNAL NAME "AVZUDTDN" LANGUAGE ASSEMBLY PARAMETER STYLE DB2SQL DETERMINISTIC FENCED NO SQL SCRATCHPAD 2048 FINAL CALL DISALLOW PARALLEL DBINFO WLM ENVIRONMENT PB1ATCB1 STAY RESIDENT YES ASUTIME NO LIMIT SECURITY USER
10:09:34:883, 21 Apr 2020 - Stop SQL Execution (Iteration Time = 26 ms).

10:09:34:883, 21 Apr 2020 - Start SQL Execution.
10:09:34:883, 21 Apr 2020 - COMMENT ON FUNCTION "TWSHAWN"."AVZW_SASAPPLICANT" (CONDITION VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, DVMNAME VARCHAR(254) CCSID EBCDIC FOR SBCS DATA, COLINFO VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, REQUEST VARCHAR(254) CCSID EBCDIC FOR SBCS DATA) IS 'DB2V:AVZW...SASAPPLICANT'
10:09:34:901, 21 Apr 2020 - Stop SQL Execution (Iteration Time = 18 ms).

10:09:34:901, 21 Apr 2020 - Start SQL Execution.
10:09:34:901, 21 Apr 2020 - GRANT EXECUTE ON FUNCTION "TWSHAWN"."AVZW_SASAPPLICANT" (CONDITION VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, DVMNAME VARCHAR(254) CCSID EBCDIC FOR SBCS DATA, COLINFO VARCHAR(24576) CCSID EBCDIC FOR SBCS DATA, REQUEST VARCHAR(254) CCSID EBCDIC FOR SBCS DATA) TO PUBLIC
10:09:34:918, 21 Apr 2020 - Stop SQL Execution (Iteration Time = 17 ms).

```

Figure 6-21 SQL Window report on the process of UDTF creation

Without a view, a DVM UDTF is addressed in SQL with a less familiar SQL syntax. UDTFs do not appear in SYSIBM.SYSTABLES and are not exposed to third-party commercial software. Creating a view addresses SQL standards and the ability to discover views that do not perform SQL Pushdown at the source and return all rows for processing to the Db2 for z/OS Subsystem.

The back-end DVM server is not technically known to the Db2 database and does not know the DVM SQL engine backing a view on a UDTF. Therefore, the WHERE clause is *not* passed through to the remote data source. Db2 receives all of the data and then applies the WHERE clause. This process is not a challenge for Db2 in many cases, but what if the virtualized table contains billions of records?

In this example, a query SELECTs from the VIEW on the UDTF:

```

SELECT TEMPID, NAME, ADDRESS, EDLEVEL, COMMENTS
FROM TWSHAWN.AVZW_VSASAPPLICANT
WHERE TEMPID >450

```

The following SQL statement is more efficient. The first parameter in the syntax for the DVM UDTF is a pair of single quotation marks with nothing in them ". This spot is reserved for a WHERE clause predicate:

```

SELECT TEMPID, NAME, ADDRESS, EDLEVEL, COMMENTS
FROM TABLE (TWSHAWN.AVZW_SASAPPLICANT ('WHERE TEMPID >450', 'AVZW...SASAPPLICANT', '5, TEMPID, NAME, ADDRESS, EDLEVEL, COMMENTS', ''))

```

Some query tools, such as IBM QMF, allow you to parameterize the SQL when host variables are used. Figure 6-22 shows the WHERE clause that is prompted for at run time.

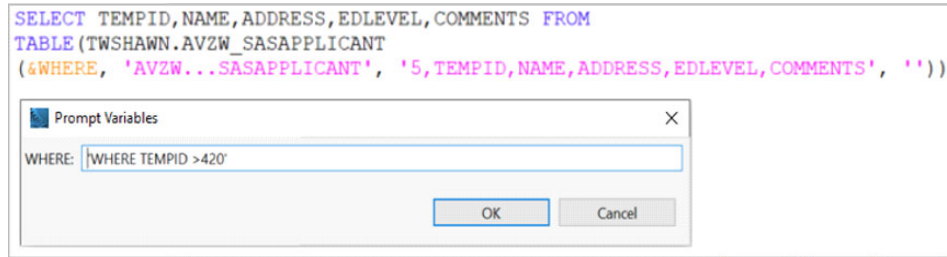


Figure 6-22 Prompting for a WHERE clause at run time

If the WHERE clause is operating on TEXT or DATE\TIME values that require single quotation marks, the single quotation marks must be doubled to two single quotation marks, not a double quotation mark, as shown in Figure 6-23.

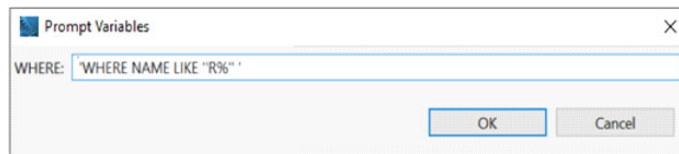


Figure 6-23 Character data in the WHERE clause

The final parameter also is a set of empty single quotation marks. This format is used to contain DVM runtime options, such as Map Reduce.

## 6.7 Db2 federation

DVM for z/OS features a level of integration with IBM's distributed Db2 family portfolio by distributing the DVM JDBC and ODBC drivers across all on-premises and cloud offerings, including:

- ▶ Db2 AESE
- ▶ Db2 Warehouse
- ▶ Db2 Warehouse on Cloud
- ▶ IBM Integrated Analytics Systems
- ▶ IBM Netezza
- ▶ Cloud Pak for Data running on-premises and Cloud Pak for Data as a Service

Drivers do not need to be downloaded or installed, and Db2 family database engines are not needed for data federation because this capability now is a built-in, including capability that does not require manual commands for creating wrappers or data type mappings.

The Db2 database optimized the DVM server as a remote database and information architecture and performs SQL Pushdown. It also can create local Nicknames or Remote Tables that map to virtual tables that are provisioned on the DVM server for underlying IBM Z sources, such as VSAM, IMS, Adabas, Sequential Files, Logstream, Syslog, SMF, and Tape systems.

Complete the following steps:

1. Connect to the database that requires access to DVM server by using the command line or the Data Studio Manager (DSM), Data Management Console (DMC), Unified Consoles as part of Db2 Warehouse on-premises and cloud offerings, and the IIAS system.
2. Create a connection server that points to the DVM server along with all other options that are specified in the following command, and in Figure 6-24:

```
db2 "CREATE SERVER <server_name> TYPE JDBC WRAPPER JDBC OPTIONS
(DRIVER_PACKAGE '<path to DVM jdbc driver>',
DRIVER_CLASS 'com.rs.jdbc.dv.DvDriver',
URL 'jdbc:rs:dv://<dvm_ip_address>:<dvm_port>;DBTY=DVS;SUBSYS=NONE;
PMDSQL=true',
pushdown 'Y',
DB2_MAXIMAL_PUSHDOWN 'Y',
db2_varchar_blankpadded_comparison 'Y',
db2_char_blankpadded_comparison 'Y',
collating_sequence 'Y')"
```

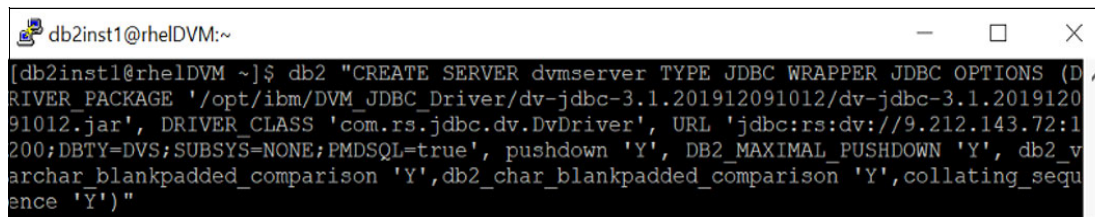
A screenshot of a terminal window titled 'db2inst1@rhelDVM:~'. The terminal shows the execution of a db2 command to create a server named 'dvmserver'. The command is: db2 "CREATE SERVER dvmserver TYPE JDBC WRAPPER JDBC OPTIONS (DRIVER\_PACKAGE '/opt/ibm/DVM\_JDBC\_Driver/dv-jdbc-3.1.201912091012/dv-jdbc-3.1.201912091012.jar', DRIVER\_CLASS 'com.rs.jdbc.dv.DvDriver', URL 'jdbc:rs:dv://9.212.143.72:1200;DBTY=DVS;SUBSYS=NONE;PMDSQL=true', pushdown 'Y', DB2\_MAXIMAL\_PUSHDOWN 'Y', db2\_varchar\_blankpadded\_comparison 'Y', db2\_char\_blankpadded\_comparison 'Y', collating\_sequence 'Y')". The output of the command is not visible in the screenshot.

Figure 6-24 Server creation command

The following options are specified in server creation statement:

- ▶ pushdown 'Y'  
The federated server considers allowing DVM evaluate operations. If you set PUSHDOWN to N, the federated server retrieves select columns from the remote data source and does not allow the DVM server evaluate other operations, such as joins.
- ▶ DB2\_MAXIMAL\_PUSHDOWN 'Y'  
The federated server pushes as many parts of the query as possible to DVM for processing.
- ▶ db2\_varchar\_blankpadded\_comparison 'Y' and db2\_char\_blankpadded\_comparison 'Y'  
The federated server pushes filtering that is based on character columns to DVM server.
- ▶ collating\_sequence 'Y'  
Character or numeric data predicates comparison, character range predicates comparison, and sort operations might be pushed down if collating sequences are the same.

Create a user mapping for all users that require access to the federated DVM server, as shown in the following db2 statement and in Figure 6-25 on page 128:

```
db2 "CREATE USER MAPPING FOR <user> SERVER <server_name> OPTIONS (REMOTE_AUTHID
'XXX', REMOTE_PASSWORD 'xxx')"
```

```
db2inst1@rheIDVM:~
[db2inst1@rheIDVM ~]$ db2 "CREATE USER MAPPING FOR db2inst1 SERVER dvmserver OPTIONS
(REMOTE AUTHID 'fbor', REMOTE PASSWORD '08quara0')"
DB20000I The SQL command completed successfully.
```

Figure 6-25 User mapping command

To validate the access to DVM server, create a nickname for a remote DVM Virtual Table or Virtual View:

```
db2 "CREATE NICKNAME <nickname> FOR <server_name>.<virtual table/view>"
```

## 6.8 IBM Cloud Pak for Data

IBM Cloud Pak for Data is a fully integrated data and AI platform that helps modernize the way data can be collected, organized, analyzed, and infused with AI. The platform is cloud native and is designed to add powerful new capabilities for data management, DataOps, governance, business analytics, and AI.

IBM Cloud Pak for Data delivers the modern information architecture to turn AI aspirations into tangible business outcomes, while improving governance and protecting your data.

IBM Cloud Pak for Data also can access z/OS data by using Data Virtualization Manager for z/OS. By using a built-in JDBC driver (for more information about the DVM-JDBC driver, see Chapter 5, “Connecting to non-Z data sources” on page 79), any z/OS data that is virtualized with DVM is available to various Cloud Pak for Data applications. It can be cataloged, analyzed, and infused with AI on the platform.

In this section, we show how to access DVM virtualized data sources by using the Cloud Pak for Data DVM connector component.

### 6.8.1 Cloud Pak for Data interface and adding a DVM connection

Complete the following steps to add a DVM connection:

1. Log in to an available IBM Cloud Pak for Data instance, as shown in Figure 6-26.

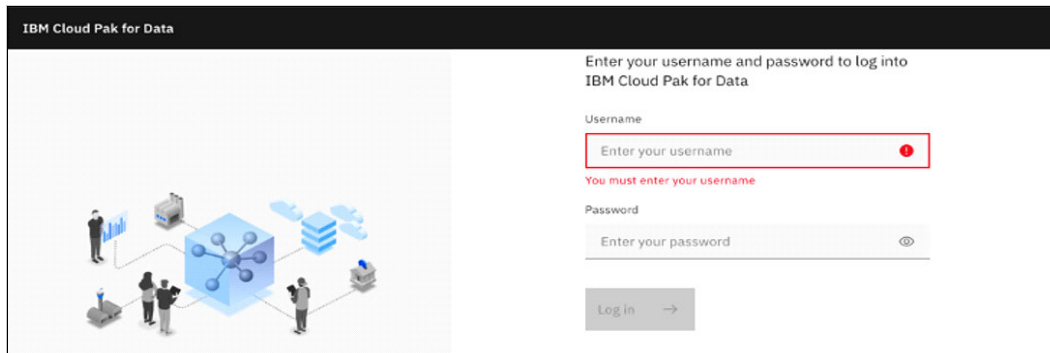


Figure 6-26 IBM Cloud Pak for Data login

2. Access the Platform connection from the main drop-down menu on the Cloud Pak for Data environment to create a connection and choose **Data Virtualization Manager for z/OS** as the connection service, as shown in Figure 6-27 and Figure 6-28.

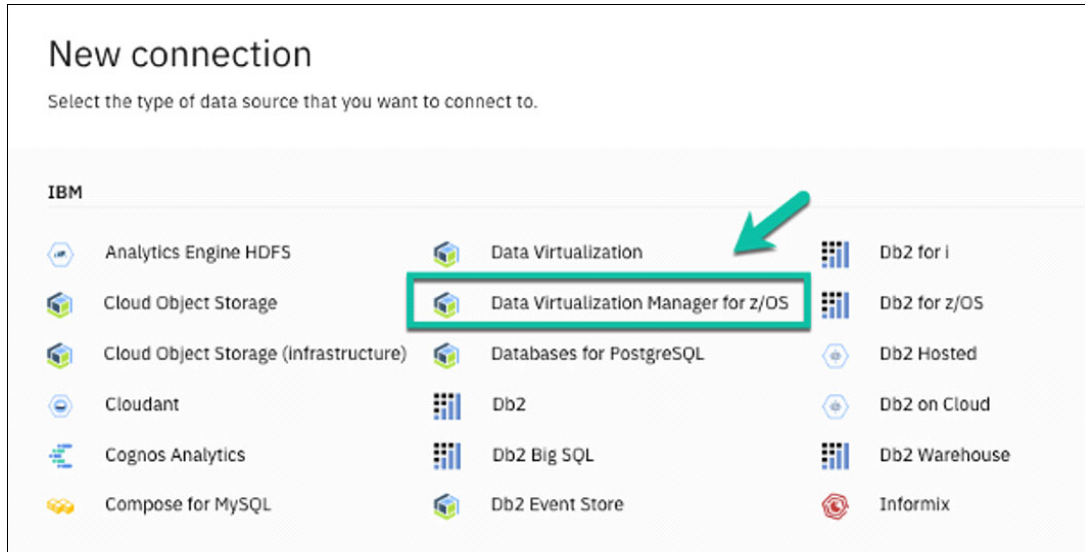


Figure 6-27 Available connection service for DVM for z/OS

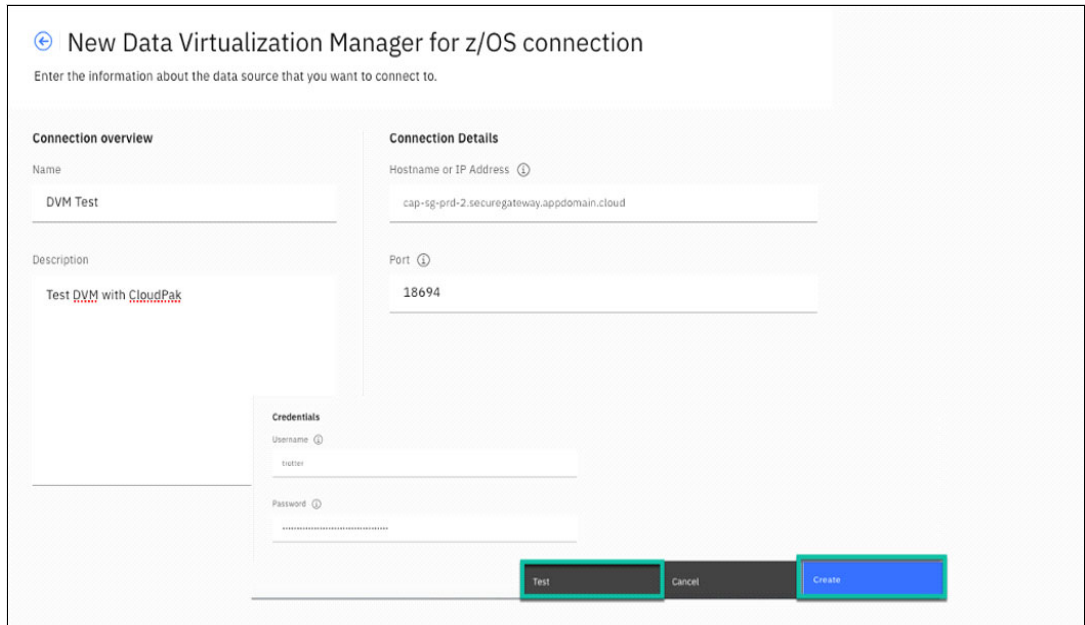


Figure 6-28 Connection service name, hostname, Port, and access credentials

## 6.8.2 Previewing data from your newly connected DVM server

After a connection is established, you can easily discover provisioned data on the DVM server. This connection provides access to persisted data on the mainframe for single queries or JOINS across sources, such as VSAM, IMS, Adabas, Sequential Files, Syslogs, Logstream, SMF, and tape. By using the SQL editor on the Cloud Pak for Data user console, you can preview, refine, and create virtual assets that map to the DVM for z/OS information architecture.

Therefore, you can create public or private projects that often are used by Data Engineers or Data Scientists and work to request the publishing of virtual assets to the Watson Knowledge Catalog by the Data Curator that is responsible for Data Governance.

After remote IBM Z data assets are published to the Watson Knowledge Catalog, they are accessible for Machine Learning, Analytics, and AI-related activities.

From the DVM Connection, you can create a project or open a project where you want to work with the data that is available. The data assets, which include individual tables, metadata, or connections, are available from within a CPD project.

Figure 6-29 shows the projects for an active user.

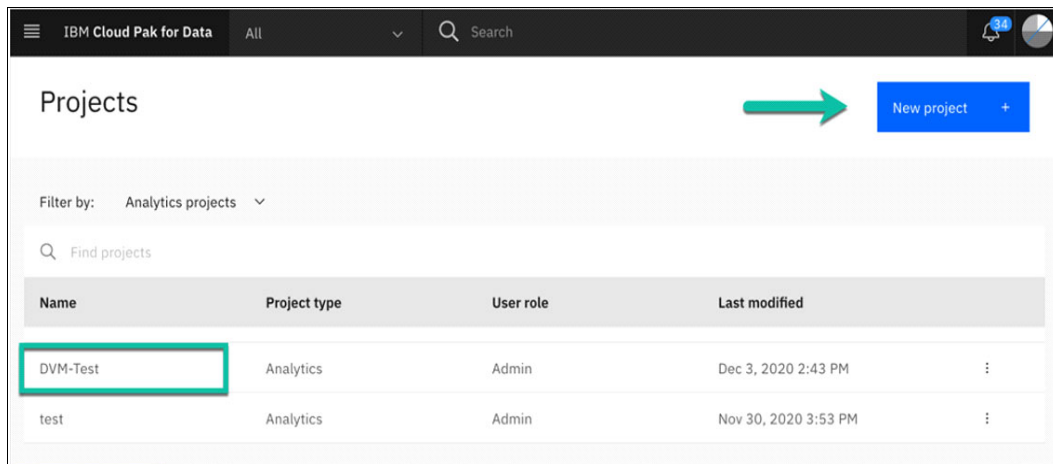


Figure 6-29 List and create projects in IBM Cloud Pak for Data



## Managing and monitoring

Data Virtualization Manager for z/OS (DVM for z/OS) supports following interfaces to the DVM server:

- ▶ Traditional interface in ISPF
- ▶ Graphical user interface (GUI) that is called DVM Studio
- ▶ Batch interface
- ▶ API interface to retrieve status information about DVM

This chapter discusses the various interfaces that can be used with the DVM server and includes the following topics:

- ▶ 7.1, “Accessing the ISPF interface” on page 132
- ▶ 7.2, “DVM Studio” on page 139
- ▶ 7.3, “Batch interface” on page 161
- ▶ 7.4, “API interface” on page 164
- ▶ 7.5, “Metadata” on page 166



## 7.1 Accessing the ISPF interface

The purpose of this interface is to monitor and administer the main functions of the DVM server by using the ISPF interface through TSO commands. TSO commands also can be incorporated into the ISPF menu.

The following command shows how to access the ISPF interface:

```
EX h1q.SAVZEXEC(AVZ) 'SUB(dvm subsystem)'
```

Where:

- ▶ h1q is the high-level qualifier of the DVM libraries of your DVM subsystem
- ▶ dvm subsystem is the name of your DVM subsystem

The command option allows access to the ISPF interface that is specific to the DVM server without being integrated into any menu. Only users that know the exact command are allowed access.

A better option is to include access to the ISPF interface into a menu option on one of the ISPF menu panels. This configuration limits access to user IDs that are allowed to use specific log on procedures. When creating a menu option in ISPF that runs a command, refer to the MVS manuals or consult your z/OS system programmer.

### 7.1.1 DVM server ISPF panel

The DVM server ISPF panel is shown in Figure 7-1.

```
IBM Data Virtualization Manager for z/OS
Option ==> _____

Interface Facilities:
1 ACI 5 IDMS
2 Adabas 6 IMS
3 CICS 7 VSAM/Sequential
4 DB2 8 DSSPUFI

SSID : AVZ1
Version : 01.01.00
Date : 20/04/24
Time : 22:50

Data Virtualization Server Administration:
A Remote User - Manage Remote Users
B Server Trace - Server Trace Facility - SIS SSID: AVZ1
C AVZ Admin. - Manage Data Virtualization Server
D Data Mapping - Data Mapping Facility
E Rules Mgmt. - Event Facility Management
F Monitor - Monitor Server Activity
G Streams - Streams Administration
H Services - Services Administration
I Instrumentation - Instrumentation Server Administration
```

Figure 7-1 Main ISPF menu of DVM

### Switching DVM servers

Throughout the main panel and the subsidiary panels, you can switch servers for some options. In the main panel, you can switch for the Interface Facilities and the Server Trace.

You switch servers by entering the correct server name in the input field and then enter the option of your choice. This process works only if the DVM server you want to switch to is actively running; otherwise, you receive an error message.



## Main panel sections

The main panel is divided into the following sections:

- ▶ Interface Facilities
- ▶ Administration

For more information about the main ISPF panel sections, see [Invoking the ISPF application using the command shell](#).

## VSAM/Sequential examples of Interface facilities

The options in the upper section give you access to the features for the various data sources. For example, you can create and maintain data mappings for VSAM and sequential files by the VSAM/Sequential option (which is option number 7).

## Server trace

The server trace option (B) on the main panel, gives you access to the log of the DVM server (see Figure 7-2).

```
----- Server Trace --- 22:51:33 24 APR 20 Cols 001 000
Command ==> _ Scroll ==> PAGE
HH:MM:SS HOST NAME -----1-----2-----3-----4-----5-----6
22:51:33 N/A FILE END EXECUTED FOR DVM.V1R1.AVZ1.SAVZMAP DD AVZMAPP, OPFB
22:51:33 N/A CALL RPC ENDED - RETURN CODE 0
22:51:33 N/A WRITE EXECUTED - SOCK 0000 - WRITE COMPLETED
22:51:33 N/A CALL RPC STARTED - MODULE DVS_SERVER
22:51:33 N/A CALL DVS_SERVER FUNCTION: GETCICSCONNECTIDS
22:51:33 N/A CALL RPC ENDED - RETURN CODE 0
22:51:33 N/A WRITE EXECUTED - SOCK 0000 - WRITE COMPLETED
22:51:33 LAPTOP-0 SELECT BUILDINFO() - SQLCODE 0
22:51:33 LAPTOP-0 SQL ENGINE HPO OPEN-CURSOR - SQLCODE 0
22:51:33 LAPTOP-0 SQL ENGINE HPO FETCH - SQLCODE 100
22:51:33 LAPTOP-0 SQL ENGINE HPO CLOSE-CURSOR - SQLCODE 0
22:51:33 N/A WRITE EXECUTED - SOCK 0000 - WRITE COMPLETED
22:51:34 LAPTOP-0 DSNRLI BYPASSED CLOSE THREAD - RC 0 REASON 00000000 SQLCODE
22:51:34 N/A SQ Closing DV engine
22:51:34 LAPTOP-0 SQL ENGINE CLOSE DATABASE - RC 0
22:51:34 N/A WRITE EXECUTED - SOCK 0000 - WRITE COMPLETED
22:51:34 N/A CANCEL AIO STARTED - SOCK 0000 - CANCEL AIO INITIATED
22:51:34 N/A CANCEL AIO EXECUTED - SOCK 0000 - CANCEL AIO COMPLETED
22:51:34 N/A CLOSE STARTED - SOCK 0000 - CLOSE INITIATED
22:51:34 N/A CLOSE EXECUTED - SOCK 0000 - CLOSE COMPLETED
***** ***** BOTTOM OF MESSAGES *****
```

Figure 7-2 DVM server trace

It shows activities, such as user log ons and log offs, query runs, and their results. It also shows errors and can be helpful in problem determination.

## Modifying the display of the Server Trace

The Server Trace display is regulated by a profile. You can change this profile by entering the word `profile` in the command line and pressing it. The panel that is shown in Figure 7-3 is displayed. Here, you can select the trace items you want to see by setting the Y/N switches, or by selecting a specific job name of the user ID, and so on.

```
----- Server Trace Profile -----
Command ==>
More: +
JOBNAME* ==> _ ==> ==> ==>
USERID* ==> ==> ==> ==>
COLOR* ==> ==> ==> ==>
CONNECT ==> ==> ==> ==>
VCID ==> ==> ==> ==>
HOSTNAME* ==> ==> ==>
TCB ==> ==>
SSID* ==> ==>
XIDTOKEN ==>
GTRIDTKN ==>
CONVTKN ==>
MSGORIGIN* ==> ==>
XTOKEN ==>
ABN => Y ADA => Y AIB => Y APM => Y ATH => N BKR => N CMD => Y CPG => Y
DET => Y DIS => Y ECI => Y ENA => Y EXC => Y FIL => Y GLV => Y IDF => Y
IMS => Y ITC => Y I/O => Y JVA => Y JVS => Y MDB => Y MFL => Y MQS => Y
OTC -> Y OTM => Y LOG => Y PUB => Y RPC => Y RRS => Y RSF => Y SIS => Y
SOM -> Y SQL => Y SQM => Y STG => N STR => Y TOD => Y TSO => Y TXT => Y
TYP => Y VPD => Y VTB => Y WLM => Y WWW => Y XCF => Y XMI => Y XTX => N
ZED => Y ZSR => Y 6.2 => Y
```

Figure 7-3 Profile panel of the Server Trace

## Data mapping

The data mapping module is where you can create and maintain the data mappings for the various data sources.

## Monitor

The monitor module (option F) gives you insight into what is happening in your DVM server. When you open option F on the main panel, the panel in Figure 7-4 is shown.

```
----- Server Activity ----- SSID: AVZ1
Option ==> _
1 Interval Activity - Server CPU Interval Summary
2 Remote Users - Monitor Remote Users
3 Remote Users by Group - Monitor Remote Users by Group
4 Storage Monitor - Virtual Storage Utilization
5 Task Monitor - Active Product Tasks
6 Statistics - Current Product Statistics
```

Figure 7-4 Main menu of the Monitoring function of DVM

### Interval activity

The first option (Interval Activity) shows CPU usage of the DVM server at 15-minute intervals. The information is spread over three panels, which can be shown by using PF11 (shift right) and PF10 (shift left). The only fixed column on the panels is the timestamp, which indicates the start time of each interval (see Figure 7-5).

```
----- Interval Summary ----- Scr 3 Row 1 of 2
Command ==> _ Scroll ==> PAGE
LCs: D Display Detail F Format P Print CB S Show CB

Interval zIIP Qual zIIP zIIP CP SRB Enclave
Start CPU Time CPU Time CPU Time CPU Time CPU Time
2020/04/24 22:45:00 .000000S .000000S .000000S .000000S .042580S
2020/04/24 22:30:00 .000000S .000000S .000000S .000000S .000000S
End
```

Figure 7-5 Resume per interval of 15 minutes

Figure 7-5 also shows how much CPU time during the interval that is qualified for offloading to zIIP engines (column “zIIP Qual CPU Time”). However, it also shows how much CPU time during the interval was off-loaded to zIIP engines (column “zIIP CPU Time”). These columns provide to the degree that the DVM server is using zIIP engines, or whether you need more zIIP capacity. The Interval Activity also shows how many users were connected during the interval (column “User Count”) and how many SQL queries were run (column “SQL Count”).

### Remote users

The second option (Remote Users) gives you the list of all user IDs that are connected to the DVM server at a specific time. You also can review the activities that a user ran (line-command T), get details about the CPU usage by the user (line-command U), cancel a thread running on the DVM server (line-command C), or disconnect the user from the DVM server (line-command K).

### Other options

Other options on the Monitor panel give you an overview of storage usage (option 4 - Storage Monitor), active tasks (option 5 - Task Monitor), and statistics about the DVM server (option 6 - Statistics).

## 7.1.2 Creating virtual tables in the ISPF interface

In this section, we look at how to create a virtual table for a flat file when the ISPF interface is used.

### Creating the source library

In DVM, the source library is the library where your copybooks are stored. If you want to create a virtual table on the non-relational data source, you need a copybook.

Open option **D (Data Mapping)** → option **S (Source Library Management)** → option **1 (Create Source Library Map)** and complete the following fields (see Figure 7-6 on page 136):

- ▶ Name: The short name for our source library within DVM.
- ▶ Description: An optional description of our copybook.
- ▶ Data Set Name: The physical name of the library in z/OS. Make sure to put the name between commas.

```

----- Source Library Definition -----
Command ==>

Name COPYBOOK
Description . . .
Replace N (Y / N) Replace an existing definition

Source from a Data Set
Data Set Name. . 'WGELDER.COPYBOOK'

Source from a Natural Library
Natural Library. _____ Natural Library name
Adabas DBID. . . _____ Database ID
Adabas File. . . _____ FUSER / FDIC file number
Server type. . . _ (C - CICS / B - Batch) Generic ACI Server to run query

```

Figure 7-6 Source library definition panel

Press Enter, and any message (failure or success) appears in the upper right when the process is done. Return to the main panel by pressing **PF3** three times.

## Creating the virtual table

Complete the following steps to create a virtual table:

1. Enter **option 7 (VSAM/Sequential)** → **option 2 (Extract Seq)**. In the first panel (see Example 7-7), complete the following information and then, press **Enter**:
  - Source Library Name: The name of our copybook library. Make sure to include the correct member within the library.
  - Start Field: The field within the data structure of your flat file where you want the data mapping to start. The mapping considers the entire structure within the copybook until the next field at the same level as the start field, unless you define another end field.
  - Map Name: This field is optional. If you leave it blank, DVM assigns the name of the first field in the mapping as the map name.

```

----- DMF Map Creation Utility ----- SSID: AVZ1
Command ==>

Source Library Name. 'WGELDER.COPYBOOK(SALESOFF)'
Start Field. SALES-OFFICE N (Case Sensitive)
* End Field. N (Case Sensitive) (Opt)
* Map Name OFFICES (Opt)
* Use Offset Zero. . . Y (Y / N)
* Convert Var to True. N (Y / N)
Flatten Arrays . . . C (C / Y / N) *See HELP for more details
* Map Data Set Name. . _____ (0 pt)

* These fields are always reset to their default settings when this panel is
 first entered

```

Figure 7-7 First panel of the mapping definition

2. In the next panel (see Figure 7-8 on page 137), enter only the DSN name and then, press **Enter**. Upon successful execution, the system returns to the previous panel with a message in the upper right.

**Note:** DSN (R) is the physical file name of the data source in z/OS (never between commas on this panel)

```

----- Sequential Extract -----
Command ==> _____

For access only via the Server, please enter the following:
Enter DSN (R) WGELDER.SALESREP.OFFICE

If DSN and/or MEMBER name(for PDS(e) data sets) are allowed to
be viewed by the client please enter a corresponding column name:
 DSN column name (0): _____
 Member column name (0): _____
Columns are allowed to be used in search criteria (Y/N) (0): _
Post Read Exit name . : _____
Pre-Write Exit name . : _____

```

Figure 7-8 Second panel of the mapping definition

3. Go back one panel by pressing **PF3** and select **option 5**. When pressing PF3, the list of maps in the metadata catalog is refreshed. When done, the message Refresh Successful appears in the upper right. Press **PF3** again to return to the main menu.
4. Use **option 8 (DSSPUFI)** to test your virtual table. This option is similar to SPUFI. On the panel, complete the following information (see Figure 7-9):
  - Change Options?: If you want to review or change the defaults for the execution of your query, you can set this option to Y. If not, you can set it to N.
  - Input Data Set: The name of a partitioned data set with a member name. The data set should exist. If the member does not exist, DVM creates it. You can use any standard 80-byte record PDS.

```

----- DSSPUFI -----
Command ==> _____

Required fields are red, optional fields are green.
More: +

General specifications:

 AVZ subsystem ID. . . AVZ1
 DB2 subsystem ID. . . _____

 Change options? . . . N (Y/N)

Input specification:

 Input data set. . . 'WGELDER.QUERIES(OFFICES)'
 Volume serial . . . _____

Report specifications (select with /, blank DSN = use temp):

 Query results . . . / DSN: _____
 Request summary . . . DSN: _____
 Request trace . . . DSN: _____
 SQLDA dump. DSN: _____

```

Figure 7-9 The input panel for testing virtual tables

5. Press **Enter**. If you set the Change Options? field to Y, the panel that includes the default settings displays. Review and change any settings as necessary and press **Enter**.  
If you set the Change Options? field to N, you do not need to press Enter again.

The member that you indicated on the previous panel opens and now can be edited. You can insert your query or queries here (close each query with a semicolon). To run immediately, you can enter three semicolons (;;;) on the command line and press **Enter**. A view-only temporary data set appears with the query results, as shown in Figure 7-10.

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
VIEW SYS20121.T003638.RA000.WGELDER.R0126754 Columns 00001 00072
Command ==> _____ Scroll ==> CSR
***** ***** Top of Data *****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000001 SELECT * FROM OFFICES
000002
000003 Result set #1
000004
000005 OFFICE_ID OFFICE_NAME OFFICE_ADDRESS
000006 =====
000007 01 BRANCH 01 BRANCH ADDRESS 01
000008 02 BRANCH 02 BRANCH ADDRESS 02
000009 03 BRANCH 03 BRANCH ADDRESS 03
000010
000011 =====

```

Figure 7-10 Output of testing a virtual table in ISPF

### Showing the new virtual table in the ISPF interface

The new virtual table can be visualized in the ISPF interface.

Enter **option 7 (VSAM/Sequential)** → **option 3 (Map Display)**. The list of data mappings is displayed (see Figure 7-11).

```

----- Data Mapping Block ----- Scr 1 Row 89 of 95
Command ==> _____ Scroll ==> PAGE
LCs: P Print Map S Show Map D Disable E Enable K Delete X Display

Structure
Name Type Status MR Language Date Time USERID Note
MTBOUED_ Enabled Y View 19/10/21 11:33 MTBOUED
MTB1_DBO Enabled Y DB2 20/02/22 19:01 MTBOUED
NEONCONV Enabled Y Catalog **/**/** 00:00 AI38KB1
X OFFICES Enabled Y Sequential 20/04/30 18:52 WGELDER
PARTRODA Enabled Y View 18/03/13 11:58 DMJBCHA
PARTROOT Enabled Y View 18/10/18 13:10 mlowe

```

Figure 7-11 List of existing virtual tables

If you enter X in front of the map, the contents of the map display, as shown in Figure 7-12.

```

----- Data Mapping Elements ----- Scr 1 Row 1 of 5
Command ==> _____ Scroll ==> PAGE
LCs: P Print Map S Show Map D Disable E Enable C Change
Map name: OFFICES
FIELD COLUMN
NAME NAME NOTE
SALES-OFFICE SALES_OFFICE
OFFICE-ID OFFICE_ID
OFFICE-NAME OFFICE_NAME
OFFICE-ADDRESS OFFICE_ADDRESS
OFFICE-CITY OFFICE_CITY
End

```

Figure 7-12 Showing the contents of a virtual table

### Displaying the new virtual table in DVM Studio

When you start DVM Studio, the newly created virtual table does not appear in the list of provisioned data objects. This issue is the result of the fact that any change that is made directly to the metadata in the DVM server is not automatically propagated to DVM Studio.

Therefore, right-click the **Virtual Tables** and select **Refresh**. The new virtual table is displayed in the list.

### 7.1.3 ISPF interface and IBM Parallel Sysplex

The ISPF interface allows you to switch between DVM servers that are running on the same LPAR. However, it is not possible to switch between DVM servers that are running on different LPARs; even when these LPARs are within an IBM Parallel Sysplex® and the DVM servers are running in data sharing mode.

## 7.2 DVM Studio

By using DVM Studio, users can create and manipulate virtual tables and views, create queries, and generate code to embed queries into applications. After it is installed, getting started with DVM Studio is a matter of a few clicks.

The interface is started from the desktop by clicking DVM Studio icon. If DVM studio and the DVM server are both running, DVM Studio automatically connects to the DVM server.

Upon your first use of DVM Studio, the server must be defined. Click **Set Server...**, or **Set Current Server**, as shown in Figure 7-13. A dialog panel displays in which you enter your credentials.

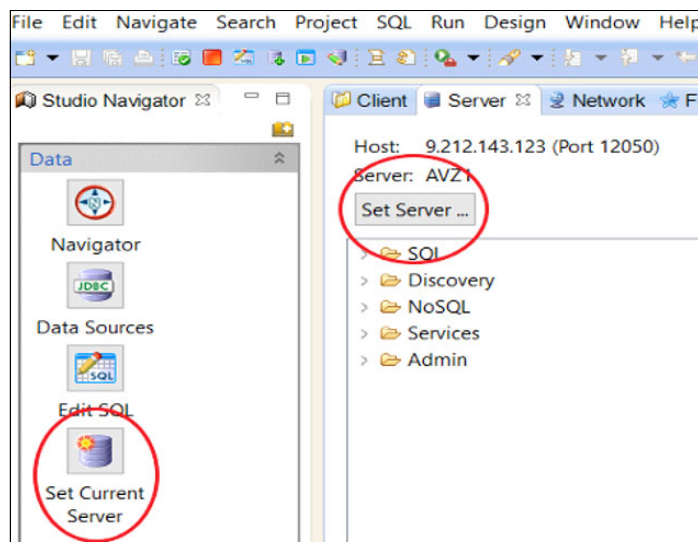


Figure 7-13 Configure the server to DVM Studio



## 7.2.1 Navigator wizard

The most useful widget in DVM Studio is the Navigator, which is the default panel that is shown when DVM Studio is started. This panel is used to create and manipulate virtual tables, virtual views, APIs, and target systems that connect source libraries and review DVM parameters.

The Navigator opens the list of virtual tables and virtual views. Various data sources on IBM Z and outside IBM Z can be accessed by clicking **SQL** → **Data**. Two entries are available: the first features the name of your DVM server; the other entry is called Other Subsystem, as shown in Figure 7-14.

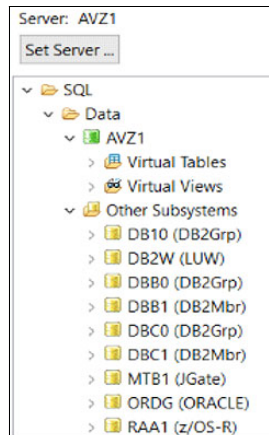


Figure 7-14 SQL Data menu of DVM Studio

The data section under the DVM server name contains all of the metadata that you created and stored directly on the DVM server, including virtual tables and virtual views.

The data section in the display that is called Other Subsystems lists the connections to all relational data sources for which objects do not require you to create a virtual table; for example, Db2 subsystems. Here, you can access the objects of these subsystems and create queries directly on these objects.

To create a view over multiple data sets, a virtual table must be created for each data source before creating a virtual view. The creation of a virtual view requires that virtual tables exist.

The Set Server section includes the following options (see Figure 7-15 on page 141):

- ▶ SQL
- ▶ Discovery
- ▶ NoSQL
- ▶ Services
- ▶ Admin



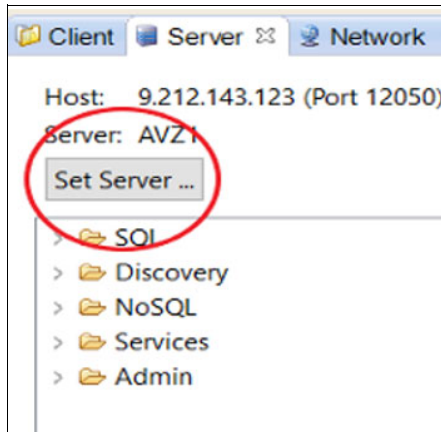


Figure 7-15 Set Server options

These options are described next.

## SQL

Clicking **SQL** opens a drop-down menu in which the following options are available:

- ▶ Data
- ▶ Storage procedures
- ▶ Target systems

Each menu option features a filtering capability that you can access by right-clicking the option and selecting **Set Tree Filter**, as shown in Figure 15. A dialog window opens in which you can enter your filtering criteria.

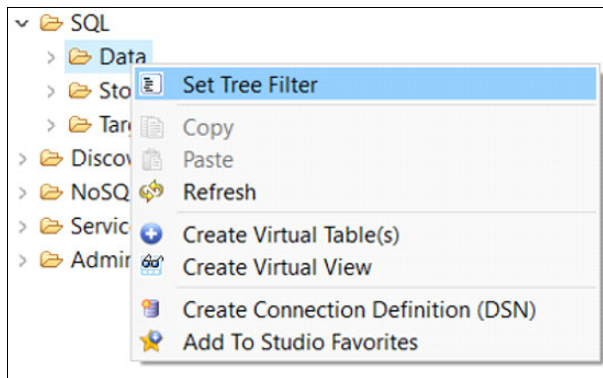


Figure 7-16 Filtering tree data is available on any menu option

The individual SQL options are described next.

### Data

SQL Data allows you to display a list of virtual tables and virtual views along with the various access paths to data.

### Stored Procedures

The next item on the main menu of DVM Studio is Stored Procedures under SQL. Here, you can create connections to stored procedures on Db2. You can also generate code on these stored procedures. However, you cannot virtualize these stored procedures; therefore, they cannot be combined with other objects within DVM Studio.

## Target Systems

Target Systems shows virtual destinations for your query outputs. Target systems can be on a DBMS or zFS (typically mounted on a directory in UNIX System Services).

All queries that are in the DVM server must include a target data source. Whenever you run a query in DVM, the query must point to a target system.

## Discovery

The Discovery option that is shown in Figure 7-17 enables access to IDMS data sources. Data discovery also provides integration with IBM Application Discovery and Delivery Intelligence (ADDI) and IBM Rational® Asset Analyzer.

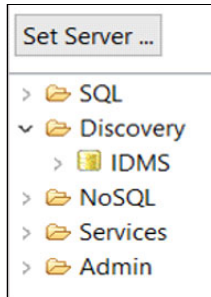


Figure 7-17 Discovery menu

## NoSQL

The NoSQL option enables users to create access to data sources that use the MongoDB language. This option is useful in environments where data scientists prefer to use MongoDB instead of SQL.

The NoSQL option is available when access is enabled for MongoDB to connect to DVM. This access allows MongoDB to generate and run queries against DVM's virtualized data.

For more information about activating this option, see Chapter 4 of the User's Guide at [IBM Documentation](#).

## Services

The Services menu includes options with which you can create, test, and administer APIs (see Figure 7-18):

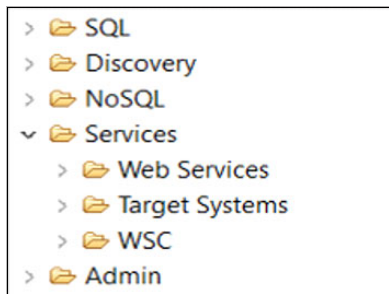


Figure 7-18 Services menu

### ► Web services

By using the Web Services option, you can create, test, and deploy APIs, along with z/OS Connect Enterprise Edition.

An API SOAP interface is handled natively by DVM. If an API uses REST it must interlock with z/OS Connect Enterprise Edition.

- ▶ Target systems

The Target Systems option is used to create the target systems that are needed for API executions. Most of the DVM server administration uses the ISPF interface. A few tasks that DVM Studio can perform also are available.

- ▶ WSC

Web Service consumption.

## Admin

Figure 7-19 shows some of the administration tasks that are available under the Admin menu:



Figure 7-19 Administration menu

- ▶ Server Parameters

The Server Parameters option allows you to look up the value parameters of the DVM server and connect source libraries with data mapping sources. This facility allows you to define which libraries are in use for data virtualization.

- ▶ Source Libraries

Source libraries point to the libraries the DVM server must access specific types of mainframe data. The source library is a server metadata object that is referenced by the DVM server. All library names, data set names, and SYSIN parameters must adapt to your local DVM installation.

- ▶ Virtualization Facility

The Virtualization Facility option gives you access to special objects of your data sources that are not common virtual tables. For example, virtualized PARTROOT DBD and PSB definitions of IMS.

## 7.2.2 DVM Studio perspectives and views

The navigator is one of the many views that is available on DVM Studio. These views are grouped into perspectives (some views appear in more than one perspective).

The DV Data perspective is by far the most important, but it is not the only one. At the upper right of your DVM Studio panel, a button is available (see Figure 7-20) with which you can access the list of perspectives. The perspectives that are open are shown to the right of the button. Clicking the perspective button opens a pop-up window in which you can select other perspectives. If you want to close a perspective, right-click the button of that perspective and choose **Close**.



Figure 7-20 Perspectives button

### Data Virtualization Data perspective

The Data Virtualization Data perspective groups views that are needed to virtualize objects on (or off) the mainframe, develop and run queries, generate code, handle APIs and so on.

### Studio Navigator panel

The options on the Studio Navigator panel focus on the views on the perspective (see Figure 7-21).

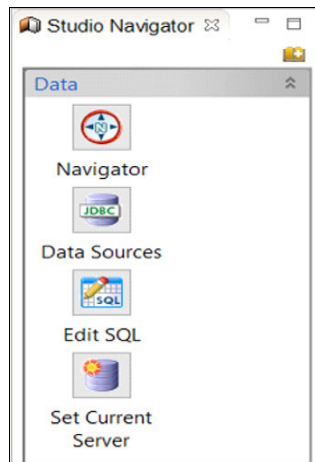


Figure 7-21 DVM Studio navigator panel

### Data Navigator panel

The Data Navigator panel features several views. The navigator view is the most important view. The navigator view gives you access to the virtualization features of DVM. This view is shown in the upper center of the GUI.

## Generated Objects panel

When you generate queries or code, it appears in Generated Objects panel. As shown in Figure 7-22, this panel is in the upper right. You now see views with extensions, such as .sql (for generated SQL statements), or .java for generated Java code.

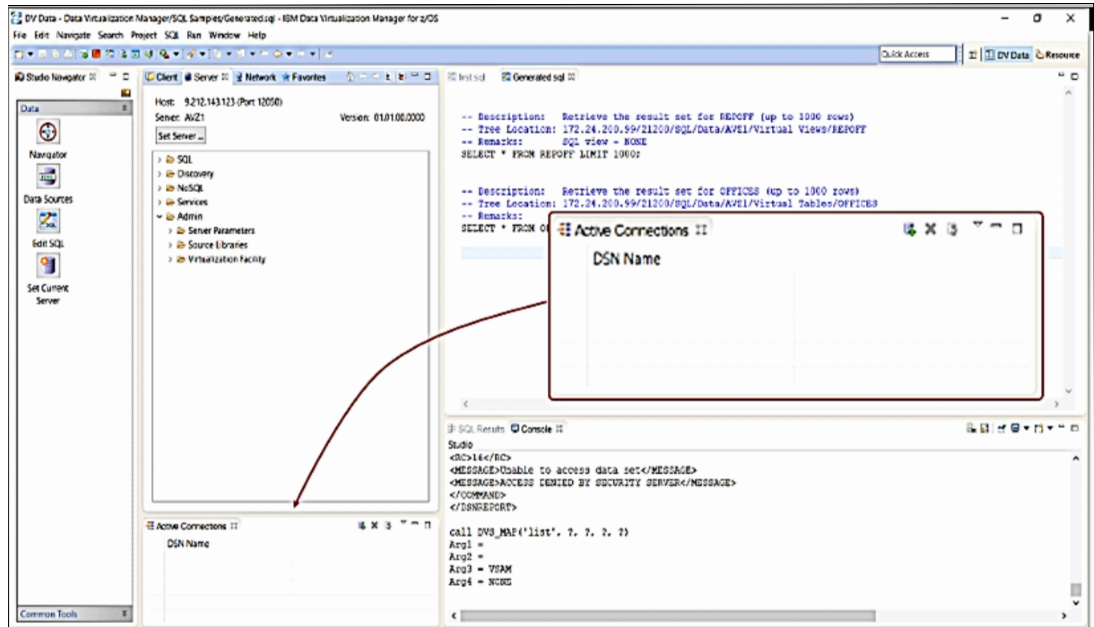


Figure 7-22 An overview of the default DV Data perspective

## Output panel

In the output panel (as shown in the lower right of Figure 7-22), you now see the results of queries (SQL Results view) or messages when you connect to a server (Console view).

## Connections panel

The Connections view shows open connections. Right-clicking the view allows you to open a new connection or close an existing connection (see Figure 7-22).

## Working with the views, panels, and perspectives

Panels and views are flexible to work with because they can be resized and moved. You also can modify perspectives.

### Modifying a perspective

You can close a view by clicking the **X** that is on the right side of the tab of the view. You can add a view to your perspective by clicking in the menu bar **Window** → **Show View** → **Other**, as shown in Figure 7-23.

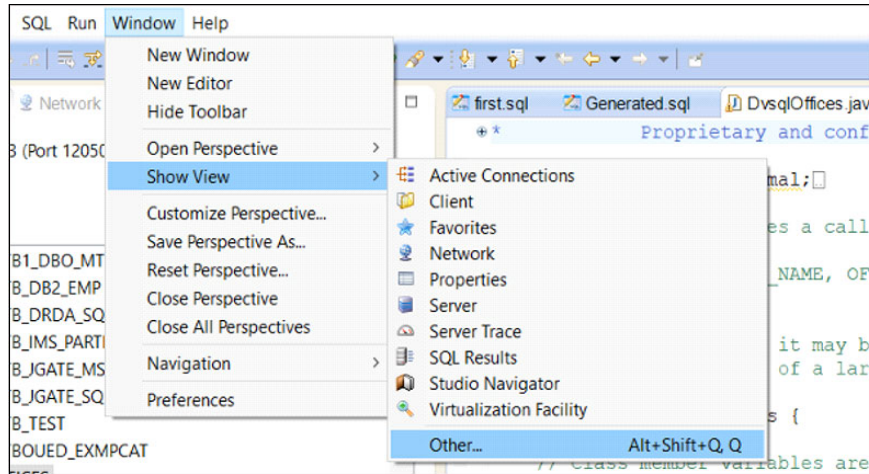


Figure 7-23 Changing the contents of a perspective

Suppose that you want to add the Debug view to your DV Data perspective. Clicking **Window** → **Show View** → **Other...** opens a pop-up window in which you see a list of perspectives and views, as shown in Figure 7-24. Select the Debug perspective and then, the Debug view.

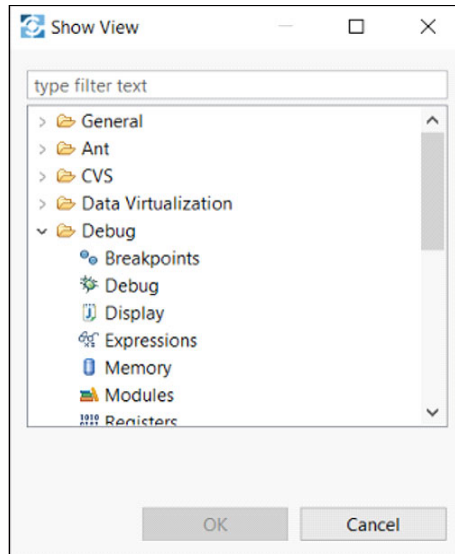


Figure 7-24 List of available of Virtual Views

When you click **OK**, the view is added to one of the panels in your perspective (in our example, the Output panel). If you want to preserve this new setting of your view, right-click the perspective icon in the upper right of your panel and select **Save as**. You can give your perspective a new name and create a perspective, or save it under the existing name and replace the current perspective.

### Default perspective

When you open DVM Studio for the first time, you notice that the DV Data perspective is the default perspective. You can assign another perspective to be the default perspective by clicking **Window** → **Preferences**. In the pop-up window, you can select the perspective of your choice and click **Make Default** (see Figure 7-25).

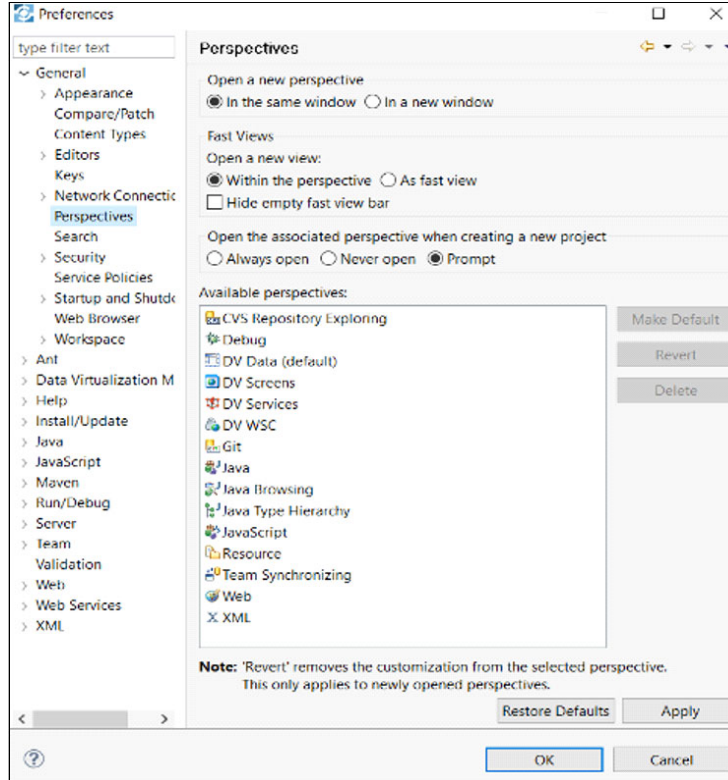


Figure 7-25 Changing the default view

### 7.2.3 Common tools

When you look at the bottom of the Studio Navigator panel, you see that another panel is available, which is called Common Tools. This panel can be opened up by clicking the double arrow on the right side (see Figure 7-26).

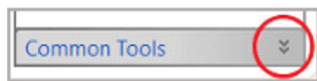


Figure 7-26 Opening the Common Tools menu

When opened, you see the Common Tools menu (see Figure 7-27 on page 148). Several useful tools are available, as described next.



Figure 7-27 Common Tools menu

## Server Trace

When you click **Server Trace**, the server trace is opened as a view in the Output panel. It is empty when it opens. Click the blue arrow to start the trace (see Figure 7-28).

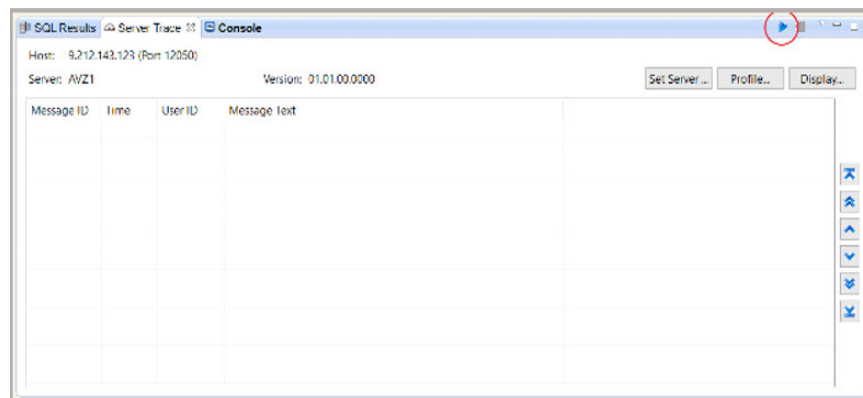


Figure 7-28 Starting the server trace

The server trace can be viewed after it is started. The arrows on the right side are used for scrolling. The upper and lower arrows navigate through the trace. The remaining two arrows scroll up and down page-by-page. The identical server trace can also display by using the DVM ISPF panel.

### **Modifying the display of the server trace**

As with the server trace on the ISPF interface, you can modify the display. In DVM Studio, click the **Profile** and **Display** buttons. In each case, a pop-up window opens in which you can modify the display, as shown in Figure 7-29.

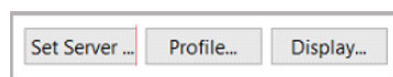


Figure 7-29 Changing the display of the server trace



On the Display panel, you can add columns to the trace display by selecting them from the list on the left and clicking **Add >** or remove them from the list on the right by clicking **< Remove**. By using the Up and Down buttons, you can modify the sequence of the selected columns on the server display (see Figure 7-30).

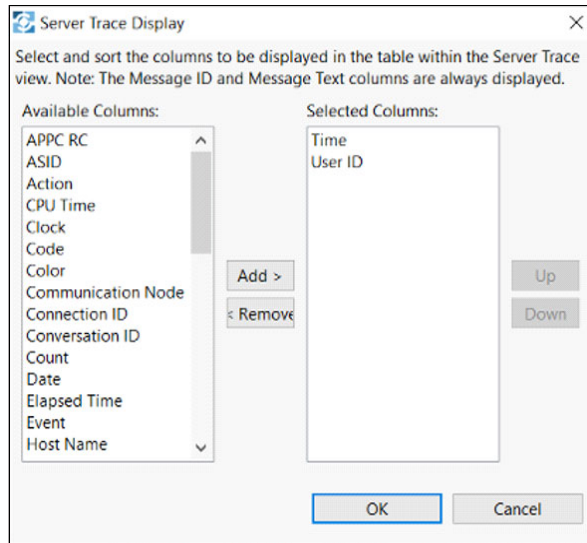


Figure 7-30 Profile pop-up panel

### Exporting the server trace

In some cases, IBM Support can request to see the server trace. You can export the server trace to an external file, which can then be sent to the Support team. Right-click any message and select **Export** (see Figure 7-31).

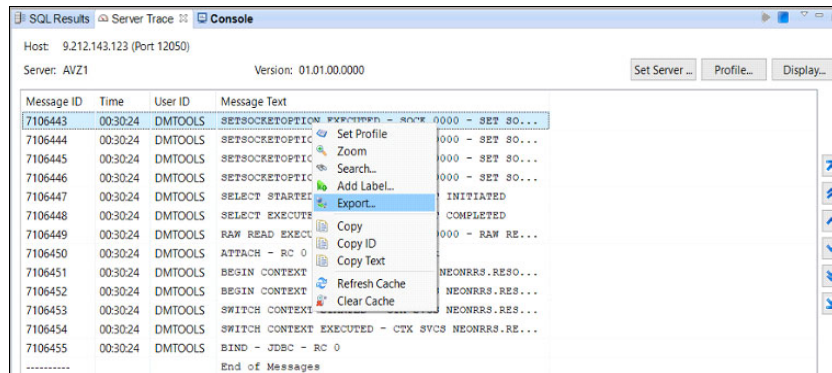


Figure 7-31 Export the server trace

A pop-up panel appears in which you enter your selection criteria and formatting information. Click **Finish** (see Figure 7-32).

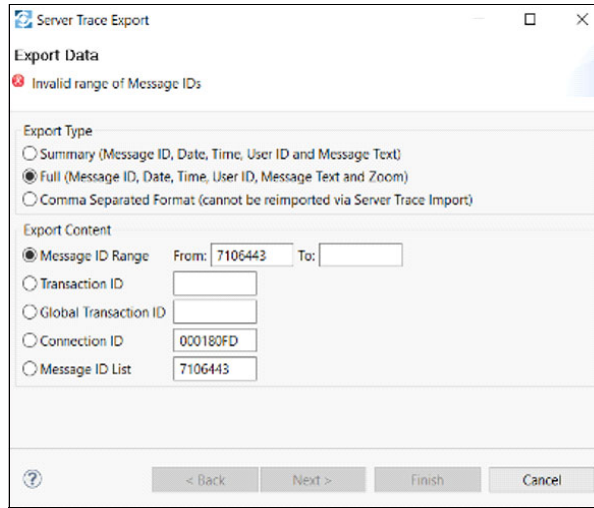


Figure 7-32 The export pop-up panel

## Gather Diagnostics

Another feature of the Common Tools menu is the Gather Diagnostics button. By clicking this button, a .zip file the includes diagnostic information is created, which can be used for troubleshooting or problem investigation. Click the button and a dialog window opens when the process completes. The dialog window also indicates where the file was stored.

## 7.2.4 More menu options

Several other menu views are available on the Studio Navigator panel. Clicking **Set Up Pages** in the upper right of the panel displays a window with other options, as shown in Figure 7-33.

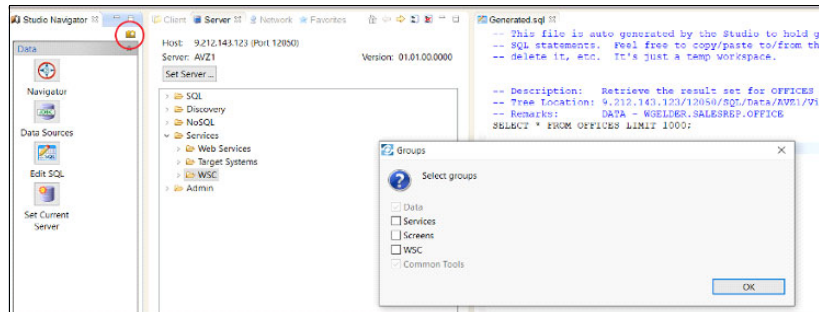


Figure 7-33 Enabling more menus

## Services

When the Services menu is enabled, a list of services is shown (see Figure 33).

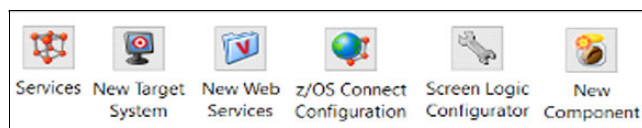


Figure 7-34 The services menu

One of the options that is available here is the z/OS Connect Configuration option. This menu option helps you configure the z/OS Connect Enterprise Edition on DVM. It also generates the necessary .xml file. When you click this option, a pop-up window opens (see Figure 7-35).

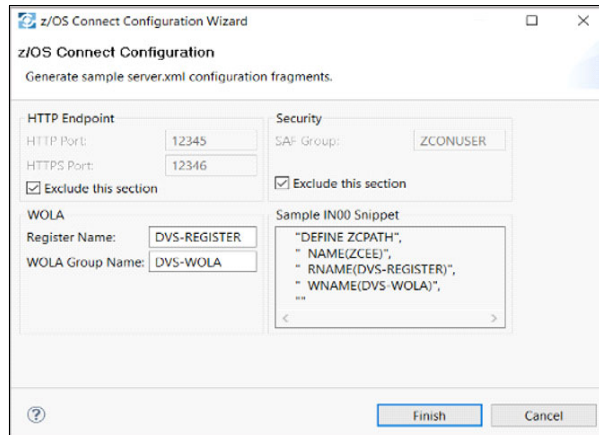


Figure 7-35 z/OS Connect configuration panel

When done, click **Finish** and the .xml appears in the generated objects panel (see Figure 7-36). At the bottom of the panel, you can toggle between the design version and the source version. The source version can be copied to the configuration file that you want to use.

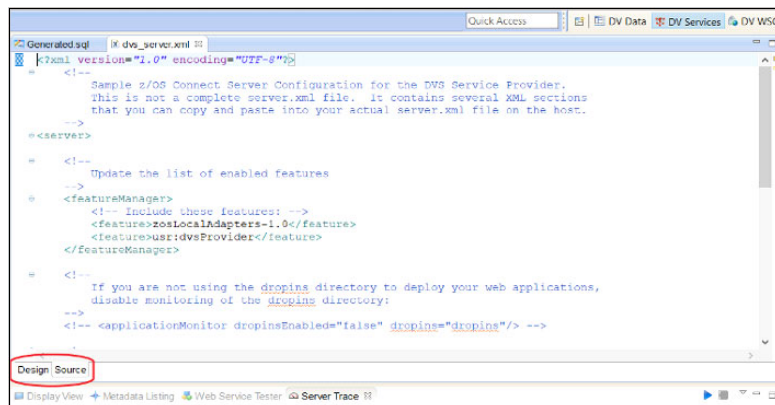


Figure 7-36 z/OS Connect configuration file 11

## 7.2.5 Using DVM Studio to virtualize IMS data segments

This section takes a closer look at virtualizing IMS data segments BY using DVM Studio. It Also demonstrates the workflow to map data and create virtual tables and views. IMS is a hierarchical database that includes specific control block mechanisms that define its data segments.

The Program Control Block (PCB) indicates which segments in the logical database the application program can process. It also indicates what type of processing the application program can perform on each segment. Internally, the PSB, PCBs, logical IMS Data Base Definition (DBD), and physical DBD are represented to IMS as control blocks. The DBD describes the name, type, and access method for the database (DEDB, MSDB, HDAM, HIDAM, HSAM, HISAM, GSAM, SHISAM, or SHSHAM).

DVM Studio uses a COBOL copybook overlay with the DBD to define fields in the IMS segment to virtualize. DVM Studio includes an explorer tree that is used to create the virtual table by right-clicking **Create Virtual Table(s)** to begin the process. Expand the explorer tree to the Virtual Tables node, right-click and then, select **Create Virtual Table**, as shown in Figure 7-37.

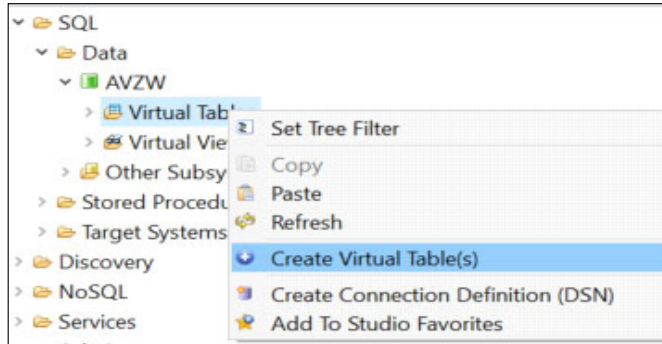


Figure 7-37 DVM Studio explorer tree filter view

You are presented a wizard to select the data set type of IMS. After it is selected, click **Next** as shown in Figure 7-38, to select an extraction type from the IMS DBD and PSB.

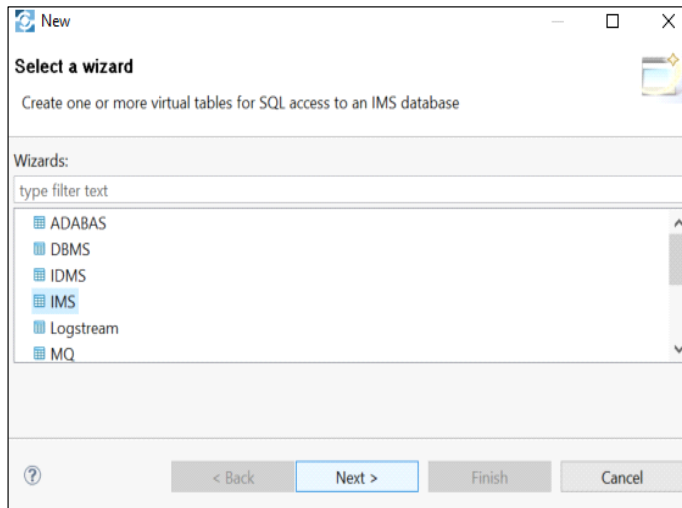


Figure 7-38 DVM Studio wizard for selecting data sets to virtualize

The process for extraction involves exporting the DBD and PSB to data sets from IMS and placing them in a well-named partitioned data set (PDS) that contain multiple members, each of which holds a separate sub-data set.

After the member is located, the data that is stored in that member must be mapped by using the DVM Data Studio in the explorer tree under the **Admin node** → **Source Libraries**.

Click **Extract DBD** on the New Virtual Table Wizard for this specific IMS segment, as shown in Figure 7-39.

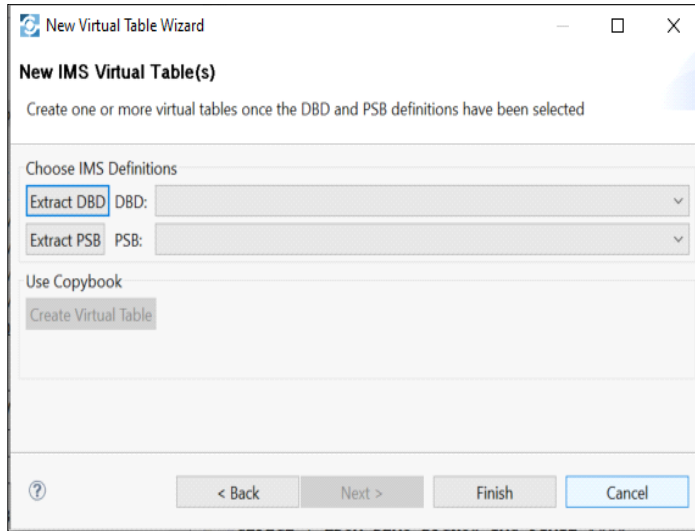


Figure 7-39 Extracting DBD using the New Virtual Table Wizard

After it is selected, the New IMS DBD Metadata Wizard starts (see Figure 7-40). This wizard is used to create the necessary metadata for the DVM server to create a logical mapping to the physical IMS data set. Details, such as the DBD name, source or target library, host system, server, and port make up the mapping that is needed to virtualize IMS segmented data into a relational format.

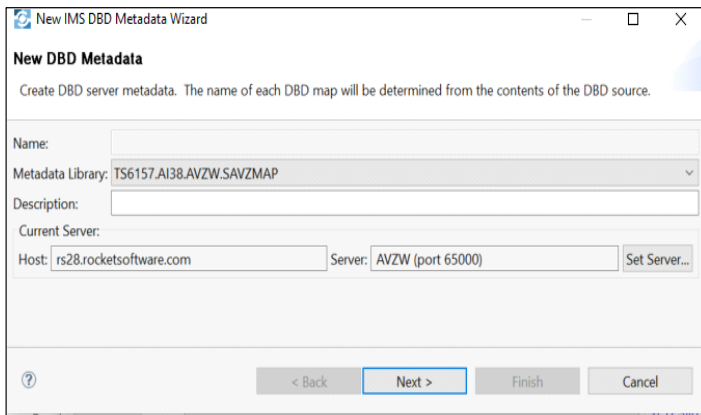


Figure 7-40 Defining DBD metadata or map for IMS segmented data

After populating the DBD metadata for the IMS segment, click **Next** to choose the suitable source library, as shown in Figure 7-41.

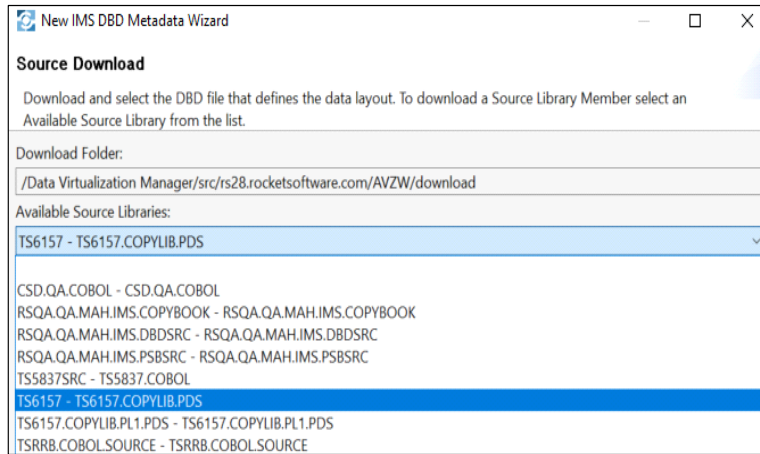


Figure 7-41 Defining the virtual source library to map DBD metadata

The wizard displays a list of source library members (DBD files) to select from that define the IMS data set layout. A DBA and system programmer might need to work together to determine the suitable library members that are targeted for virtualization. You can then review the contents of a source library before clicking **Finish**, as shown in Figure 7-42.

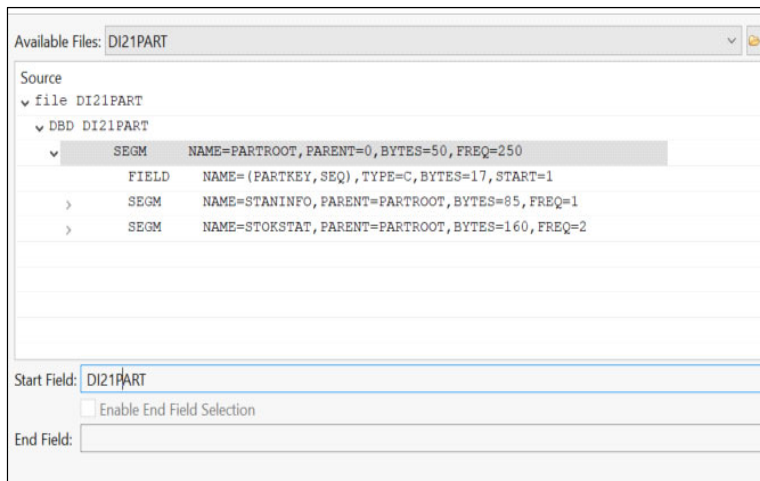


Figure 7-42 DBD wizard allows you to view the source library contents

If the virtual source library is downloaded, you can select the DBD and PSB IMS definitions directly from the drop-down lists on the New Virtual Table Wizard. Figure 7-43 shows a completed view for extracted DBD and PSB copybook.

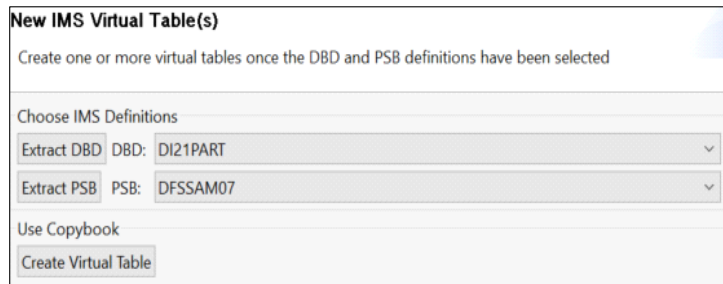


Figure 7-43 Drop-down lists on the New Virtual Table Wizard

Click **Create Virtual Table** to name the new virtual table (see Figure 7-44). A best practice is to use a name that conforms to your local standards. The name of this new virtual table is DEMO\_IMS\_BACK03. More fields are available for a description and methods to access and retrieve data. After the fields on this dialog are completed, click **Next**.

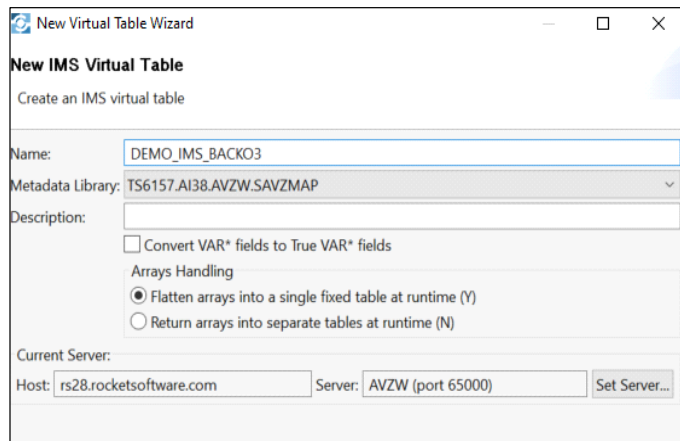


Figure 7-44 Naming the virtual table

Now that the metadata for the IMS segment is detailed and a source library is configured, you can download the suitable copybook from a list, as shown in Figure 7-45.

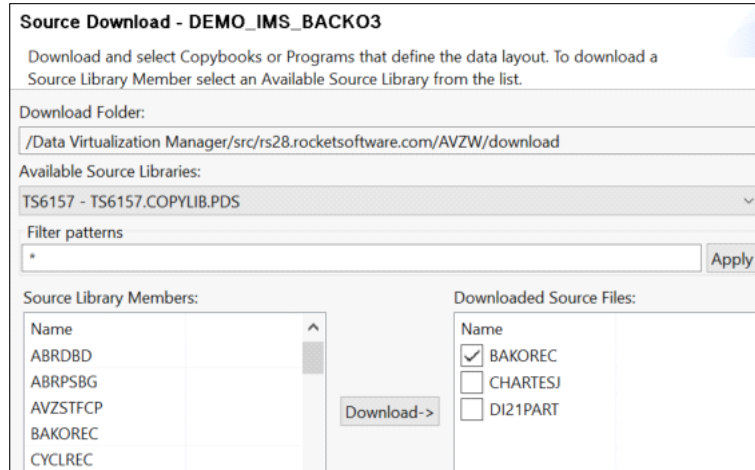


Figure 7-45 Download the source library member for the IMS data segment

Next, review the virtual table layout for accuracy, as shown in Figure 7-46.

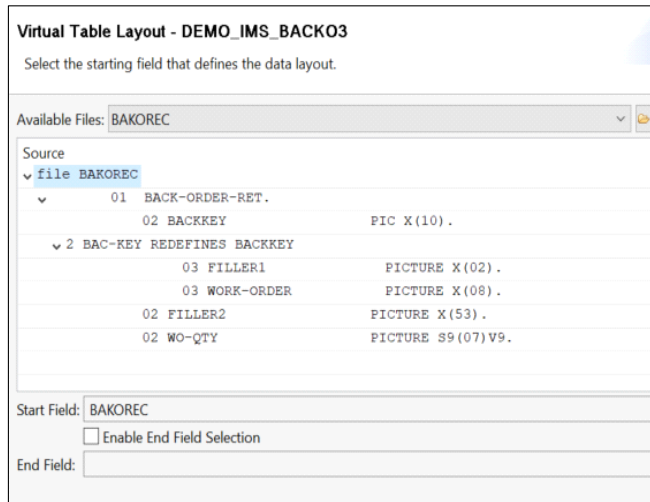


Figure 7-46 Virtual table layout

If there is nothing to redefine, click **Next**, select the target IMS segment, and then, choose an access method to the data, as shown in Figure 7-47. For IMS, two options are available: DBCTL or IMS-Direct.

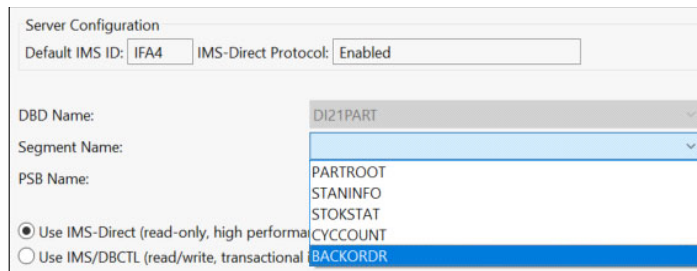


Figure 7-47 Select IMS segment and access method



Click **Finish** to create the new virtual table that is named DEMO\_IMS\_BACK03. The virtual table is now in the DVM server list of available virtual tables. Client applications can now access that data as though it were a relational database that uses standard SQL or RESTful APIs.

You can also use DVM Studio to run queries against the newly created virtual table. By default, after a virtual table is created, it is preselected in DVM Studio's explorer tree for testing. Right-click the new virtual table (in our example, DEMO\_IMS\_BAK03) and select **Generate query with \*** (see Figure 7-48).

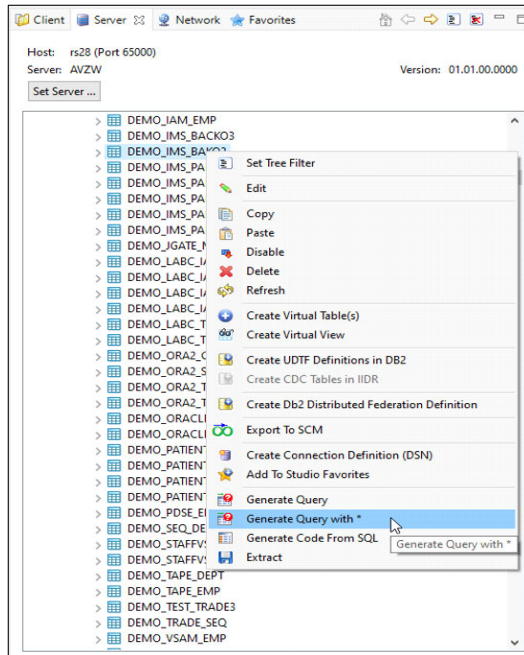


Figure 7-48 Explorer view with new virtual table preselected

This selection creates and issues a `select * from DEMO_IMS_BAK03`; SQL statement and displays the result set in DVM Studio, as shown in Figure 7-49.

|    | BACKKEY    | FILLER1 | WORK_ORDER | FILLER2 | WO_QTY | RECORD_ID     | PARENT_ID     |
|----|------------|---------|------------|---------|--------|---------------|---------------|
| 0  | 30PR237942 | 30      | PR237942   |         | 210.0  | 02JAN1N976B   | 0025509126    |
| 1  | 30PR265943 | 30      | PR265943   |         | 100.0  | 02250236-001  | 0025900326    |
| 2  | 30PR347921 | 30      | PR347921   |         | 200.0  | 02250236-001  | 0025900326    |
| 3  | 30PR426134 | 30      | PR426134   |         | 300.0  | 02250236-001  | 0025900326    |
| 4  | 30S0536609 | 30      | S0536609   |         | B34... | 023003806     | 0025900326    |
| 5  | 30S0536610 | 30      | S0536610   |         | B34... | 023003806     | 0025900326    |
| 6  | 30PR149329 | 30      | PR149329   | 1303603 | 0485   | B24...        | 027618032P101 |
| 7  | 30PR149376 | 30      | PR149376   | 1303603 | 0485   | B24...        | 027618032P101 |
| 8  | 30PR153096 | 30      | PR153096   | 1303603 | 0485   | B24...        | 027618032P101 |
| 9  | 30PR153098 | 30      | PR153098   | 1303603 | 0485   | B24...        | 027618032P101 |
| 10 | 30PR169566 | 30      | PR169566   | 1303603 | 0485   | B24...        | 027618032P101 |
| 11 | 30PR135640 | 30      | PR135640   | 0485    | B33... | 027736847P001 | 0025900326    |

Figure 7-49 Generated query results for DEMO\_IMS\_BAK03

Also, DVM Studio can be used to create virtual views over virtual tables. This feature is helpful when performing JOIN operations over heterogeneous data or when running nested operations against the same table.

Create a View can simplify query access. Views help to model data from different sources and can be queried directly or by using dynamic SQL.

After virtual tables are created and discoverable, creating virtual views is greatly simplified, because the data sets to virtual source libraries does not need to be extracted or mapped. The virtual table data assets already exist. With virtual tables, the process for creating a virtual view is similar; that is, by using DVM Studio explorer tree view, right-click **Virtual View** and select **Create Virtual View**, as shown in Figure 7-50.

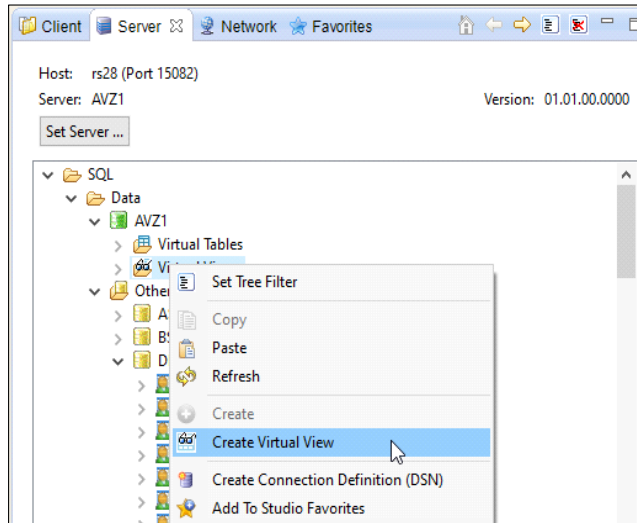


Figure 7-50 Create a virtual view using DVM Studio

After right-clicking **Create Virtual View**, a pop-up dialog appears. It is here where you provide a name for the view and select the associated library where the data exists. Click **Next**, as shown in Figure 7-51.

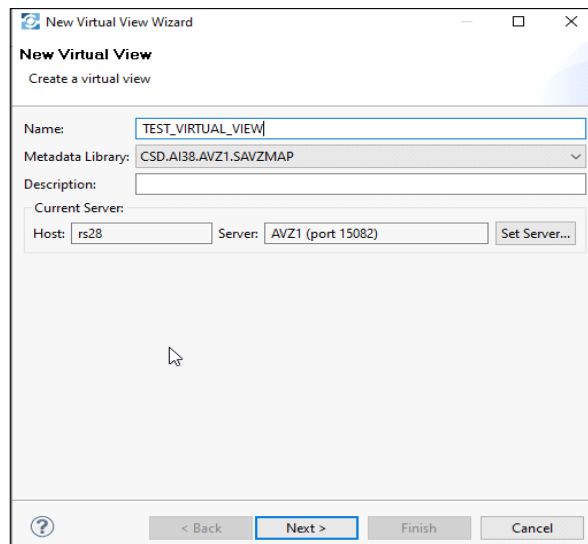


Figure 7-51 Name the new virtual view in the New Virtual View Wizard

Expand the table list using the virtual view wizard, choose the virtual table to be used for the new view and click **Next**, as shown in Figure 7-52. Change the Select Statement as wanted by using the wizard. You can remove columns you do not want and add where predicates to limit results.

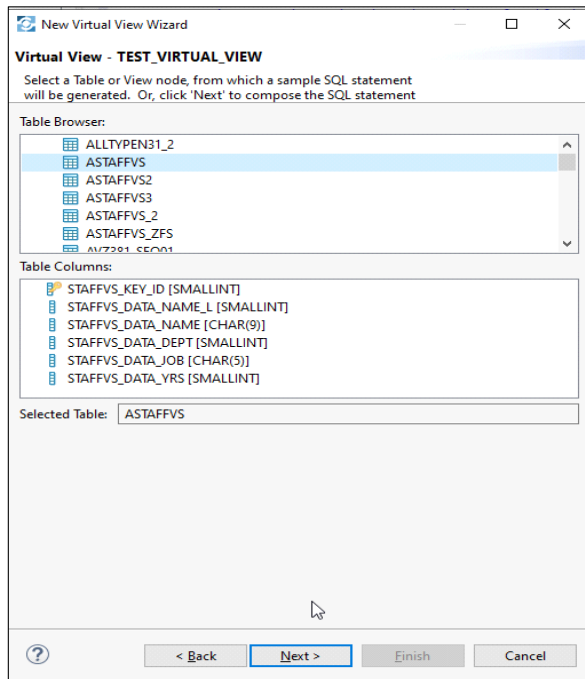


Figure 7-52 Select one or more virtual tables that make up the new virtual view

After this process is complete, click **Next** and validate the SQL by clicking **Validate**, as shown in Figure 7-53.

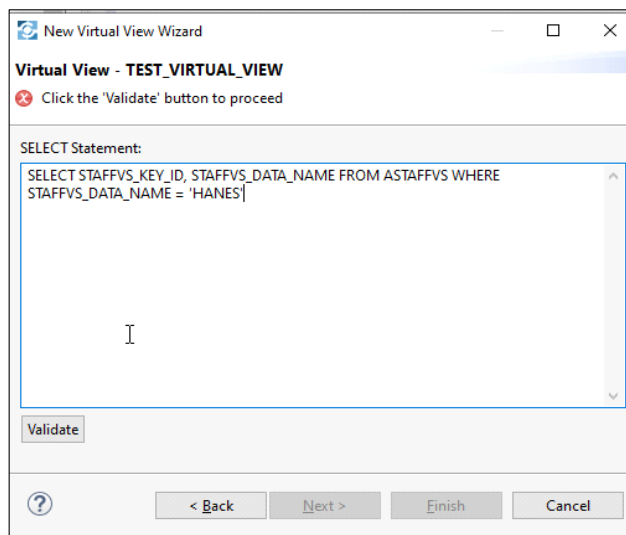


Figure 7-53 Validate the SQL statement for the new virtual view definition.

After the validated process completes successfully, click **OK** and then, click **Finish**, as shown in Figure 7-54.

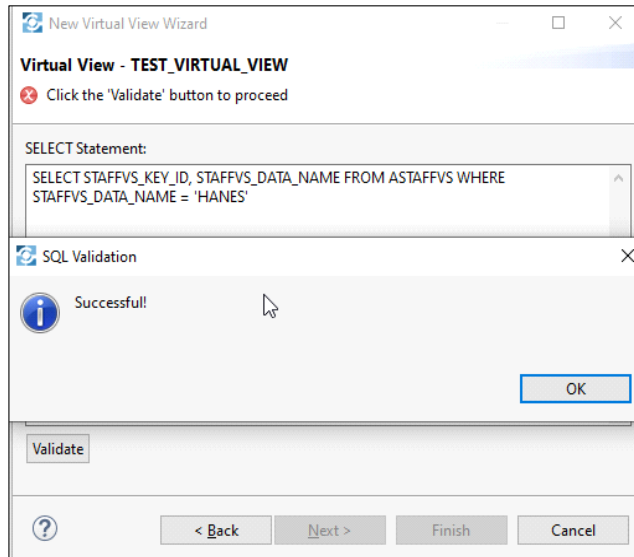


Figure 7-54 Successful SQL validation

DVM Studio returns you to the SQL server tree with the new virtual view displayed. After the new virtual view appears in the list, you can right-click the view name and select **Generate Query** to run a live test against the underlying data for its virtual table, as shown in Figure 7-55.

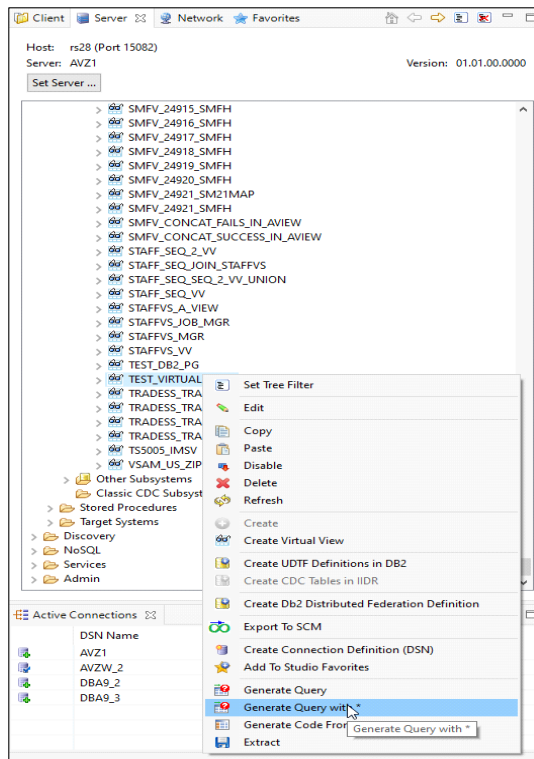


Figure 7-55 Generate a query against the newly created virtual view

Results are displayed in the SQL editor (see Figure 7-56).

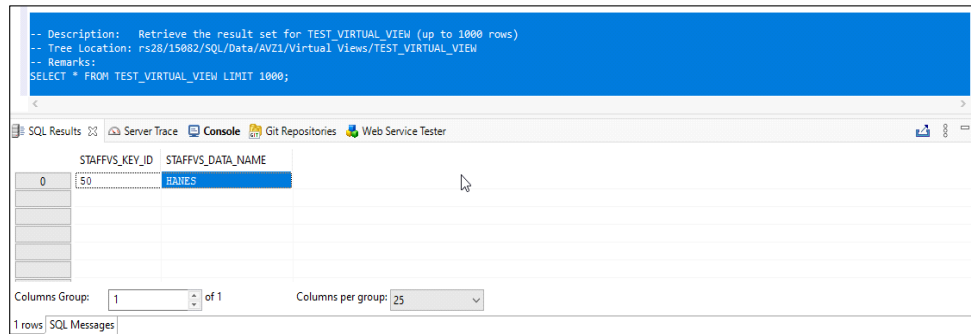


Figure 7-56 Query results from newly created Virtual View STAFFVS

## 7.3 Batch interface

The batch interface of DVM is a z/OS-based set of JCL jobs that can be used for any of the following table activities:

- ▶ Create a table mapping.
- ▶ Unload or load table mappings at the DVM server.
- ▶ Query virtual tables and views.

Jobs also are available to manage the DVM configuration and metadata, which are used during upgrades or version migration.

### 7.3.1 Creating a virtual table with the batch interface

The batch interface to create a table mapping in the DVM server can be used in situations where DVM Studio is not available or where connection issues exist. Creating a table mapping by using batch scripting is done with one job, hlq.SAVZCNTL(AVZMFPAR). The job can be used for any type of data mapping. The needed parameters are explained in the comment section of the JCL.

The job to create the same virtual table that we used earlier (OFFICES) resembles the JCL that is shown in Example 7-1. A job card (work item) must be added. Also, all library names, data set names, and SYSIN parameters must adapt to your local DVM installation.

*Example 7-1 Example JCL to create a virtual table by using a batch JCL*

---

```
// SET LOADLIB=DVM.V1R1.SAVZLOAD
// SET REXXLIB=DVM.V1R1.SAVZEXEC
//DMFEXTR1 EXEC PGM=IKJEFT01,PARM=('AVZMBTPA 0'),REGION=0M /
/STEPLIB DD DISP=SHR,DSN=&LOADLIB /
/SYSEXEC DD DISP=SHR,DSN=&REXXLIB
//SOURCE DD DISP=SHR,DSN=WGELDER.COPYBOOK(SALESOFF) /
//SYSTSPRT DD SYSOUT=* //SYSTSIN DD DUMMY
//SYSIN DD *
 SSID = AVZ1
 FUNCTION = STOD
 SOURCE = WGELDER.COPYBOOK(SALESOFF)
 START FIELD = SALES-OFFICE
 MAP NAME = OFFICES
```

```
SEQ FILE = WGELDER.SALESREP.OFFICE
/*(in our example 2 'WGELDER.DVM.DVMEXTR')
```

---

The following SYSIN parameters are used:

- ▶ SSID: The name of the DVM server.
- ▶ FUNCTION: The functions to run (in our example, STOD, which is the function to create a VSAM/Sequential table mapping).
- ▶ SOURCE: The name of the copybook library and the member for this specific mapping.
- ▶ START FIELD: The field within the data structure of your flat file where you want the data mapping to start. Usually, this field is the first field within the structure. The mapping considers the entire data structure within the copybook until the next field at the same level as the start field, unless you define another end field.
- ▶ MAP NAME: An optional field. If you leave it blank, DVM assigns the name of the first field in the mapping as the map name.
- ▶ SEQ FILE: The physical file name of the data source you want to map.

### 7.3.2 Migrate virtual tables with the batch interface

A batch interface can extract data mappings from a DVM server and store them in an XML file. The file contents are uploaded into another DVM server. It is a two-step process. The first job h1q.SAVZCNTL(AVZGNMPM) performs the following tasks:

1. Generates the job that is used to load the extracted data mappings into the target DVM server.
2. Creates the extracted file.

Next, you run the generated job to the load on the target DVM server. You can specify more than one map on the extract if wanted, but be aware that wild cards cannot be used. You must individually list all of the maps that you want to migrate by using a comma-separated list that uses their exact matching names.

The .xml file that is shown in Example 7-2, WGELDER.DVM.DVMEXTR, does not need to exist. It is created automatically, if necessary. The same is true for the member name that holds the load JCL (in our example DVMEXTR). This load job will be created in the JCLLIB library included in the example JCL.

*Example 7-2 Example JCL to migrate an existing virtual table*

---

```
// SET LOADLIB=DVM.V1R1.SAVZLOAD
// SET REXXLIB=DVM.V1R1.SAVZEXEC
// SET SKELLIB=DVM.V1R1.SAVZSLIB
// SET ISPF='ISP'
// SET JCLLIB=WGELDER.JOBLIB
//*
//JCLBLD EXEC PGM=IKJEFT01,DYNAMNBR=200
//STEPLIB DD DISP=SHR,DSN=&LOADLIB
//SYSEXEC DD DISP=SHR,DSN=&REXXLIB
//SYSPRINT DD SYSOUT=*
//ISPPROF DD DISP=(NEW,DELETE,DELETE),DSN=&&PROF,
// DCB=(RECFM=FB,LRECL=80,DSORG=PO),UNIT=SYSDA,
// SPACE=(CYL,(1,1,2))
//ISPMLIB DD DISP=SHR,DSN=&ISPF..SISPMENU
//ISPPLIB DD DISP=SHR,DSN=&ISPF..SISPPENU
```

```

//ISPTLIB DD DISP=SHR,DSN=&ISPF..SISPTENU
//ISPSLIB DD DISP=SHR,DSN=&SKELLIB
//ISPLOG DD SYSOUT=*,DCB=LRECL=133,RECFM=FB
//ISPLIST DD SYSOUT=*,DCB=LRECL=133,RECFM=FB
//ISPFILE DD DSN=&JCLLIB,DISP=SHR <=JCL LIBRARY OUTPUT
//*****
//* SAY STATEMENTS WRITTEN TO THIS DDNAME
//*****
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=256,BLKSIZE=25600)
//SYSTSIN DD * PROFILE NOPREFIX ISPSTART PGM(AVZIMEX) PARM(PROGRAM(AVZMFMIG)
ARG('O') + SUBSYS(AVZ1) MAXEDQ(1000)) /*
//SYSIN DD *
 SOURCE AVZ SSID = AVZ1
 TARGET AVZ SSID = AVZ2
 TARGET LOADLIB = DVM.V1R1.SAVZLOAD
 TARGET EXECFB = DVM.V1R1.SAVZEXEC
 MAP EXPORT PDS = WGELDER.DVM.DVMEXTR
 JCL MEMBER NAME = DVMEXTR
 JCL MEMBER REPLACE = YES
 MAP = OFFICES JOBCARD =
//WGELDERA JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=H, JOBCARD1 =
// REGION=OM,NOTIFY=&SYSUID JOBCARD2 =
//* SAVE OPTION = SAVE REFRESH OPTION = REFRESH /*
//

```

---

The extracted job resembles the JCL that is shown in Example 7-2. Again, a valid job card must be added and all library names, data set names, and SYSIN parameters must be adapted to your local DVM installation.

The following SYSIN parameters are used:

- ▶ SOURCE AVZ SSID is the source DVM server from which you are extracting the data mapping.
- ▶ TARGET AVZ SSID is the target DVM server where you want to upload the data mapping.
- ▶ TARGET LOADLIB is the load library of your target DVM server.
- ▶ TARGET EXECFB is the EXEC library of your target DVM server.
- ▶ MAP EXPORT PDS is the library that holds your extracted maps. One member for each extracted map is created.
- ▶ JCL MEMBER NAME is the member name on the JCLLIB library that contains your load JCL.
- ▶ JCL MEMBER REPLACE: If the value is YES, the member with the load JCL replaces any member with an identical name in the JCLLIB library. If the value is NO and a member with an identical name exists in the JCLLIB library, the extract JCL fails.
- ▶ MAP is the name of the maps to be extracted.
- ▶ JOBCARD is the first line of the job card to be added to your load JCL.
- ▶ JOBCARD1 is the second line of the job card to be added to your load JCL (you can specify up to three jobcard lines).
- ▶ SAVE OPTION: If SAVE, the new MAP is saved on the target DVM server, even if a map exists with an identical name, REPLACE replaces it and NOSAVE discards it.
- ▶ REFRESH OPTION: If YES, the map list on the target DVM server is refreshed.

### 7.3.3 Querying virtual tables with the batch interface

You can also use the batch interface to query virtual tables and views. The query job resembles the JCL that is shown in Example 7-3. A job card must be added. Also, make sure the correct library names and DVM server names (parm SSID) are used. The SQL output is in the file that is labeled FMT. The same virtual table that we created in 7.1.2, “Creating virtual tables in the ISPF interface” on page 135 is used in Example 7-3.

*Example 7-3 Example JCL to query an existing virtual table*

---

```
//STEP01 EXEC PGM=AVZXMAPD,PARM='SSID=AVZ1,MXR=0,MXP=999'
//STEPLIB DD DISP=SHR,DSN=DVM.V1R1.SAVZLOAD
//RPT DD SYSOUT=*,OUTLIM=250000 <== SUMMARY
//FMT DD SYSOUT=*,OUTLIM=250000 <== SQL RESULT
//TRC DD SYSOUT=*,OUTLIM=250000 <== TRACE
//DUMP DD SYSOUT=*,OUTLIM=250000 <== SQLCA
//IN DD DDNAME=SYSCNTL
//SYSCNTL DD *
SELECT * FROM OFFICES;
```

---

## 7.4 API interface

The API interface of DVM can be used to retrieve the following type of information regarding the DVM address space:

- ▶ Status
- ▶ Control parameters
- ▶ Performance characteristics
- ▶ Resource consumption information

### 7.4.1 API interface purpose

DVM maintains diagnostic information regarding activities and status for active connections from JDBC, J2CA, and ODBC applications on the DVM server. This information is of the following types:

- ▶ Real-time data exists while a connection is active. Much of the information (such as CPU time and session elapsed time) changes continuously over time.
- ▶ Trace information is stored in an in-memory data set. The size of the data set sets the limit to the amount of trace information that is kept.

For system management tasks that are related to the execution status of DVM, the `getConnectionInfo` functions return information from the real-time information features. For system management tasks that are related to the diagnosis of past systems events, the `getMessages` functions return information from the trace data set.



## 7.4.2 Calling the API interface

API interface command uses the following syntax:

```
CALL DVS_SERVER('parameter', ['optional parameters', ...])
```

This command returns a standard JDBC (or ODBC) result-set. In the background, each API is a stored procedure that runs on z/OS within the DVM server. The JDBC or ODBC adapter of DVM is used to run the API call. The following parameters are needed in the connection string for the API call:

- ▶ Hostname
- ▶ Port
- ▶ UserID
- ▶ Password

We recommend that you do not to put more parameters on the connection string. All results are returned as string fields. Several APIs return information about the other APIs.

## 7.4.3 API functions

The following APIs are available:

- ▶ CALL DVS\_SERVER ('ping')
- ▶ CALL DVS\_SERVER ('getloadbalance', 'all')

The all parameter is optional and returns information about all the DVM servers within the system. When the all parameter is omitted, only information about the DVM servers within the same group is returned. The group in this context refers to the high available configuration (this terminology is similar to Db2 Data Sharing).

- ▶ CALL DVS\_SERVER('getconnectioninfo', ['usefieldnames'])

When the extra parameter usefieldnames is used, the field names are used as column headers on the result set.

- ▶ CALL DVS\_SERVER ('getmessages', ['search vector(parameter)'])
- ▶ CALL DVS\_SERVER ('getevents', ['search vector(parameter)'])
- ▶ CALL DVS\_SERVER ('getformat', 'cbbk')
- ▶ CALL DVS\_SERVER('getparametervalues', 'keyword', 'value')

The keyword is the name of the parameter.

- ▶ CALL DVS\_SERVER('setparametervalue', 'parm', 'value')

Sets the parameter 'parm' to the value 'value'.

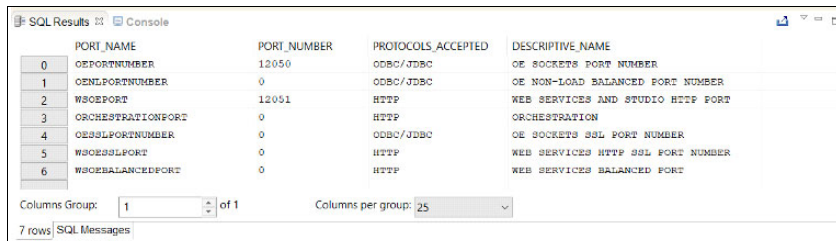
- ▶ CALL DVS\_SERVER('getparameterdescriptions')
- ▶ CALL DVS\_SERVER('getgroupnames')

This API returns the group name for the associated DVM server if the DVM server is part of a high availability configuration.

- ▶ CALL DVS\_SERVER('getdb2subsysids')
- ▶ CALL DVS\_SERVER('getimssubsysids')
- ▶ CALL DVS\_SERVER('getcicsconnectids')
- ▶ CALL DVS\_SERVER ('getportids')

## 7.4.4 API interface and DVM Studio

The API interface also can be used and tested by using DVM Studio. You need only to write call `DVS_SERVER('getportids')` in the generated.sql view, select the statement, and then, press **PF5** to run it. The results are shown in the SQL Results view (see Figure 7-57). By using the `setparametervalue` API, you can dynamically manipulate the DVM parameters from within DVM Studio.



The screenshot shows a window titled "SQL Results" with a "Console" tab. It displays a table with the following data:

|   | PORT_NAME         | PORT_NUMBER | PROTOCOLS_ACCEPTED | DESCRIPTIVE_NAME                  |
|---|-------------------|-------------|--------------------|-----------------------------------|
| 0 | OEPORTNUMBER      | 12050       | ODBC/JDBC          | OE SOCKETS PORT NUMBER            |
| 1 | OENLPORTNUMBER    | 0           | ODBC/JDBC          | OE NON-LOAD BALANCED PORT NUMBER  |
| 2 | WSOEPRT           | 12051       | HTTP               | WEB SERVICES AND STUDIO HTTP PORT |
| 3 | ORCHESTRATIONPORT | 0           | HTTP               | ORCHESTRATION                     |
| 4 | OESSLPORTNUMBER   | 0           | ODBC/JDBC          | OE SOCKETS SSL PORT NUMBER        |
| 5 | WSOSSLPORT        | 0           | HTTP               | WEB SERVICES HTTP SSL PORT NUMBER |
| 6 | WSOEBALANCEPORT   | 0           | HTTP               | WEB SERVICES BALANCED PORT        |

Below the table, there are controls for "Columns Group" (set to 1 of 1) and "Columns per group" (set to 25). At the bottom, it indicates "7 rows" and "SQL Messages".

Figure 7-57 Running the API interface from DVM Studio

## 7.5 Metadata

The metadata for this technology is contained in the catalog of the objects you created within the DVM server (mostly virtual tables and views). This catalog works similar to a database catalog, such as Db2.

Therefore, you can query this catalog as is done with any other virtual table or view in DVM. A system or database administrator can query this metadata by using DVM Studio, the ISPF interface, or batch scripting.

The following metadata tables exist in the DVM server:

- ▶ SQLENG.COLUMNS
- ▶ SQLENG.COLUMNPRIVS
- ▶ SQLENG.ERRORMSGs
- ▶ SQLENG.FOREIGNKEYS
- ▶ SQLENG.PRIMARYKEYS
- ▶ SQLENG.ROUTINES
- ▶ SQLENG.SPECIALCOLS
- ▶ SQLENG.STATISTICS
- ▶ SQLENG.TABLES
- ▶ SQLENG.TABLEPRIVS



# Performance tuning and query optimization

The Data Virtualization Manager (DVM) server optimizes the allocation and use of resources for processor types (General and zIIP) and system memory. Every resource affects the environment in how it is used when running various of workloads.

This chapter details various techniques for managing performance across the DVM server, parallelism, zIIP utilization, and query execution.

This chapter includes the following topics:

- ▶ 8.1, “Introduction” on page 168
- ▶ 8.2, “Combined GP and zIIP consumption” on page 168
- ▶ 8.3, “Parallel I/O and MapReduce” on page 170
- ▶ 8.4, “Virtual Parallel Data” on page 171
- ▶ 8.5, “Workload management” on page 172
- ▶ 8.6, “ODBC performance” on page 185
- ▶ 8.7, “Integrated Data Facility and DS Client API” on page 185
- ▶ 8.8, “Query optimization and performance” on page 186

## 8.1 Introduction

Parallel I/O, MapReduce capability, block-fetch, and memory caching all require memory to obtain optimal performance for workloads. For example, the maximum number of parallel threads that is possible for a DVM server depends on the number of zIIP specialty engines that is available in combination with available system memory.

Similarly, block-fetch of data into system memory requires adequate allocation of cache to improve run time in accessing data in memory versus more I/O cycles that are associated with data retrieval from disk.

A general slide rule applies between zIIP specialty engines and general processors. DVM throttles processing between two types of processors, whereby zIIP engine processing is restricted. An increase of processing naturally occurs on the general processors. To reduce the MSU consumption for the system, your environment must ensure that you have adequate zIIP engines to shift workloads and reduce the overall costs for processing.

The best recommendation for an initial installation is to focus on a standard resource allocation of two zIIP engines and 32 Gigabytes of memory for a 1 - 2 General Processor configuration.

Starting with a balanced resource plan simplifies monitoring resource allocation that uses SMF72 record types. The Resource Group data section of the SMF 72 record provides information about MIN/MAX capacity across various resource groups:

- ▶ Percentage of:
  - LPAR share
  - Processor capacity
- ▶ Memory limit
- ▶ MSU/H

## 8.2 Combined GP and zIIP consumption

As workloads are introduced through the DVM server, adjustments to resources can be made to allocate processing and memory to ensure optimal performance. Tests that were conducted at the IBM Systems Benchmark Center in a controlled environment demonstrate how increasing numbers of zIIP engines directly affect parallelism. Performance improves significantly with reduced elapsed time to run workloads, as shown in Table 8-1 and Table 8-2 on page 169.

**Note:** These results are from a unique test environment and are for illustration purposes only. The actual results are specific to your environment.

Table 8-1 zIIP engine use

| Server | Total CPU Time | Total zIIP Time | Total IICP Time | Total zIIP NTime | %zIIP eligible |
|--------|----------------|-----------------|-----------------|------------------|----------------|
| DVM1   | 7099.33        | 5609.55         | 1389.58         | 5609.55          | 98.59%         |

Table 8-2 Performance with parallelism and zIIP engine use

| Test | GPPs | Number of zIIP engines | Degree of parallelism | Elapsed time (ms) | SMT |
|------|------|------------------------|-----------------------|-------------------|-----|
| 1    | 8    | 0                      | 0                     | 118.96            | 1   |
| 2    | 8    | 5                      | 0                     | 98.68             | 1   |
| 3    | 8    | 5                      | 4                     | 27.05             | 1   |
| 4    | 8    | 5                      | 8                     | 17.14             | 1   |
| 5    | 8    | 5                      | 8                     | 20.84             | 2   |
| 6    | 8    | 5                      | 10                    | 17.00             | 2   |
| 7    | 8    | 5                      | 16                    | 15.73             | 2   |
| 8    | 8    | 8                      | 8                     | 13.83             | 1   |
| 9    | 8    | 8                      | 8                     | 17.62             | 2   |
| 10   | 8    | 8                      | 16                    | 11.72             | 2   |

This test was run against an older z13 machine that uses 800 Gigabytes of financial data. The test achieved approximately 99% offload to zIIP specialty engines. However, tests 2 and 4 were run against identical system resources where the degree of parallelism is set to a value of 8. This test result in a reduction of elapsed time from 98.68 minutes to 17.14 minutes.

Increasing the number of zIIP specialty engines from 5 to 8 further reduced the overall elapsed time to 13.83 minutes.

Increasing the number of zIIP engines and the degree of parallelism within the DVM server can result in performance improvements up to 1,000% for elapsed times.

**Note:** Consider the following points:

- ▶ The improvements that are noted here are from a unique test environment and are for illustration purposes only. The actual results are specific to your environment.
- ▶ Begin with smaller projects as you deploy and use the DVM server. Over time, slowly expand the use of more resources, such as memory, zIIP processing, and general purpose processing, as your deployments expand in the amount of data and number of users that are accessing the data.

## 8.3 Parallel I/O and MapReduce

DVM for z/OS optimizes performance by using a multi-threaded z/OS-based runtime engine that use parallel I/O and MapReduce capabilities to simultaneously fetch data from disk or memory. The DVM server parallelism that uses MapReduce is shown in Figure 8-1.

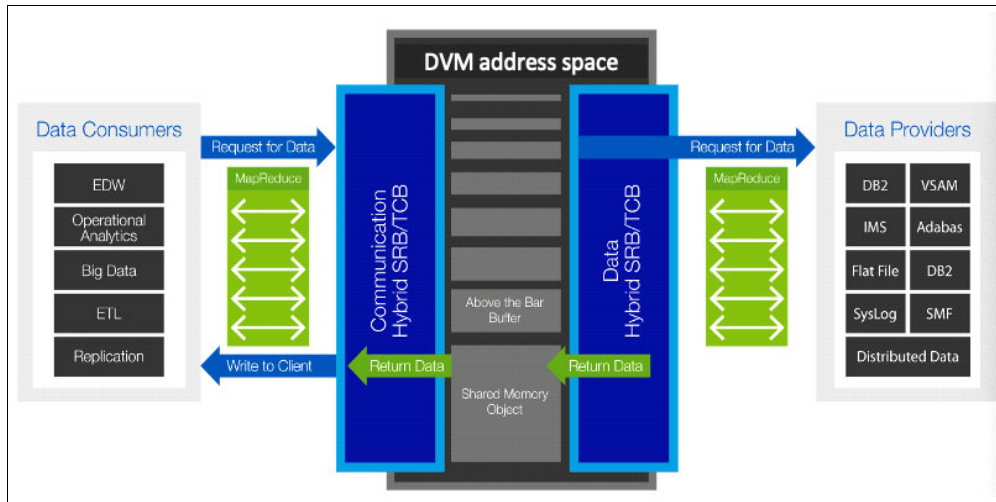


Figure 8-1 DVM server parallelism achieved using MapReduce

MapReduce reduces query elapsed time by splitting queries into multiple threads that read large files in parallel. Each interaction, whether started by a distributed data consumer or application platform, runs under the control of a separate z/OS thread. A thread is a single flow of control within a process.

The DVM server can deliver predicate pushdown and filtering to all data sources and supports heterogeneous JOINS with associated pushdown processing of filters for subqueries and their respective sources.

## 8.4 Virtual Parallel Data

Virtual Parallel Data (VPD) provides a means to cache data into defined members for faster and more optimal subsequent queries. Similar to other cache mechanisms, data is initially populated and persisted as a materialized view that can be accessed repetitively by client applications and refreshed as needed to maintain currency for the business. This cache benefits by performing disk I/O once to populate the cache, which reduces associated expense (see Figure 8-2).

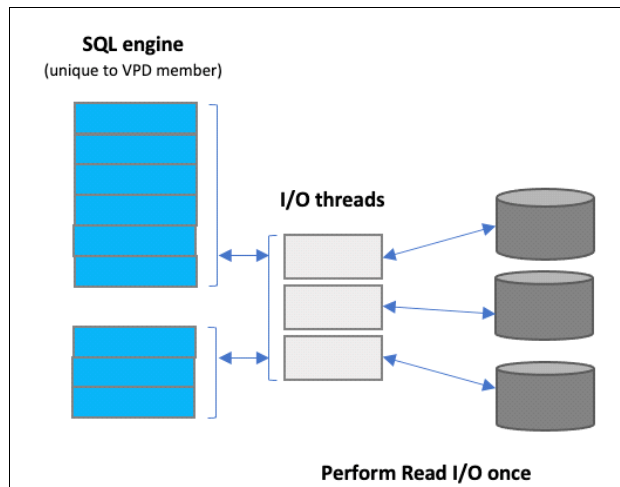


Figure 8-2 DVM VPD

VPD allows applications or users to group multiple simultaneous requests against the same data source in parallel without subsequent data refreshes. This function also allows single or multiple requests to run with asymmetrical parallelism, which separately tunes the number of I/O, client, or SQL engine threads.

### 8.4.1 Using VPD groups

All requests that are submitted against the same DVM server instance must refer to a group name. VPD groups also have a predefined amount of time for the group to persist. One or more I/O threads read the data source and then write to a circular buffer that is assigned to the VPD group. Group members share buffered data without having to read directly from the disk.

I/O threads are started when the VPD group is created, and data flows to the buffer. If the buffer fills before the group is closed, the I/O threads wait. After the group is closed and active members use the data, the buffer space is reclaimed, and I/O continues.

### 8.4.2 Example

In this example, we have a large sequential file that must be used by multiple applications. We want this file to be read only once for operational or performance reasons. A VPD group can be established for this file. The first query loads the data into cache and subsequent queries can attach to this VPD group, each specifying their own degree of parallelism.

Table 8-3 lists supported data sources and client access methods that support VIPA.

Table 8-3 Supported data sources and client access using VIPA

| Supported data sources                                                                                                                                                                                        | Supported client access                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▶ Adabas</li> <li>▶ IDMS</li> <li>▶ IMS</li> <li>▶ VSAM</li> <li>▶ Logstreams</li> <li>▶ IBM MQ Series</li> <li>▶ Sequential</li> <li>▶ Tape</li> <li>▶ zFS</li> </ul> | <ul style="list-style-type: none"> <li>▶ Batch</li> <li>▶ DSSPUFI</li> <li>▶ JDBC</li> <li>▶ ODBC</li> <li>▶ IzODA (Spark Scala, Spark Java, Python DB API)</li> <li>▶ Db2 Query Manager Facility</li> <li>▶ IDAA Loader</li> </ul> |

### 8.4.3 Considerations and limits

When considering the use of VPD, include the following information in your decision making:

- ▶ If VPD is not used, each client must create a separate virtual table to access the same set of data from the source. With VPD, DVM creates one virtual table for accessing the data by all the clients.
- ▶ The end-to-end read operation must be carried out by each client application. With VPD, the data is read from the source once and the same data is used by each client application in parallel.
- ▶ Input devices, such as tape, can be read-only serially; therefore, parallelization is not possible. Each of the client applications must read the data from source to destination serially. With VPD, DVM can read the data from tape serially once and the client applications can access the data in parallel from DVM buffers.
- ▶ Client applications that must read data in a specific order must read the data from an original data source in its specific order. Even when the data can be read from the source in parallel, VPD allows the data to be read into DVM buffers in parallel. Client applications can then read the data from buffers in any specific order.

## 8.5 Workload management

Workload management is critical to ensure optimal performance across different workloads. Performance goals for various workloads can be defined by using the IBM Workload Manager for z/OS (WLM) with the ability to further assign the required importance of each workload in business terms. Resources are assigned to specific work items to determine the ability to attain goals through continuous monitoring where resource adjustments are made to achieve wanted business objectives.

IBM Z resources are assigned based on goals that are defined in the IBM Workload Manager shown in Figure 8-3.

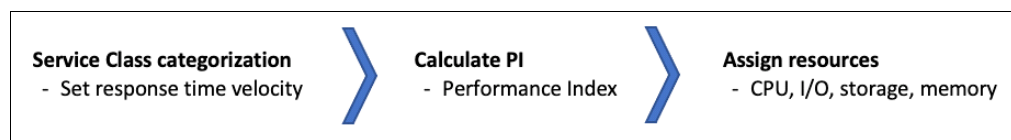


Figure 8-3 Resource priority assignments by WLM



Business priority can be specified for the WLM services in the DVM server. Specific service classes are used to inform the operating system of specific performance goals and priority levels, as shown in Figure 8-4.

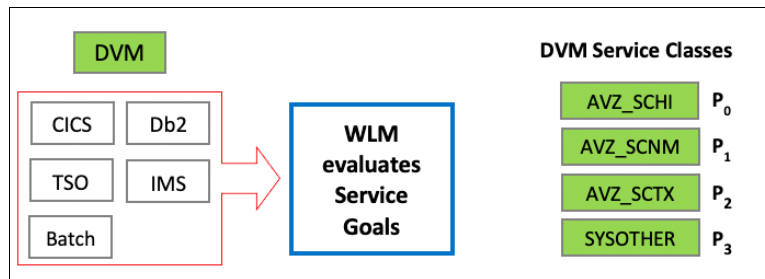


Figure 8-4 Resource priority assignments by WLM

In the service class, you assign each goal and its relative importance and associate the service class with a specific workload and resource group. The DVM server uses the following service classes:

- ▶ AVZ\_SCHI ZIIPCLASS=AVZ High priority. This service class is for IBM Data Virtualization Manager for z/OS critical work. Assign this class goal as close to SYSSTC as possible.
- ▶ AVZ\_SCNM ZIIPCLASS=AVZ Normal work. This class is for IBM Data Virtualization Manager for z/OS administrative work. Assign this class the same priorities as those that are used for DB2 master or the IMS control region.
- ▶ AVZ\_SCTX ZIIPCLASS=AVZ Client work. This service class is for client requests. Assign this class the same goals as those that support the data source for the CICS, IMS/TM, or DB2 WLM address space.

To enable the WLM policy for the DVM server, the DVM user ID (default: AVZS) can have UPDATE access to MVSADMIN.WLM.POLICY. If the WLM policy is not defined for the DVM server, WLM assigns the lowest service class SYSOTHER to DVM workloads, which negatively affects the DVM server performance.

### 8.5.1 Configuring WLM for the DVM server

DVM provides the ability to dynamically set WLM information within the AVZSIN00 configuration member by enabling the following parameters:

```

If DoThis then
 do
 "MODIFY PARM NAME(WLMFORCEPOLICY) VALUE(YES)"
 "MODIFY PARM NAME(WLMTRANNAME) VALUE(APPLNAME)"
 "MODIFY PARM NAME(WLMUSERID) VALUE(AVZS)"
 End

```

The DVM server's WLM definitions can also be specified within WLM policies. For more information, see [this web page](#).

## 8.5.2 Working with multiple DVM servers

To handle more workloads and ensure organizational service level objectives, more DVM servers can be instantiated and the server workload can be balanced across multiple DVM servers. Load balancing allows inbound connections to be automatically directed to the DVM server instance that features the most available resources for the number of connections. The overall availability of virtual storage (less than or greater than 16 Megabytes reference point) determines which DVM server instance handles an individual request.

### Managing Workload within a single LPAR

Load balancing is transparent to the client application. Client applications use a port number to connect to a DVM server, which then performs Port sharing to route a request to a more optimal DVM server as needed. TCP/IP's SHAREPORT or SHAREPORTWLM is the recommended approach to load balance workload across multiple DVM servers within a single LPAR.

### Managing Workload over multiple LPARs, Regions or Sysplex

When balancing workload across a Sysplex, Dynamic Virtual IP Addressing (DVIPA) can be used as it provides workload balancing and failover for applications that use TCP/IP services. With SHAREPORT, SHAREPORTWLM, and DVIPA, all the balancing is done at the TCP/IP layer and the Server is unaware that any balancing is occurring. The load balancing of CICS regions is handled differently in DVM servers by using LOADBALGROUP parameter.

### Using SHAREPORT and SHAREPORTWLM

Port sharing is a method to distribute workloads for DVM servers within a z/OS LPAR. TCP/IP allows multiple listeners to listen on the same combination of port and interface. Workloads that are destined for this application can be distributed among the group of DVM servers that listen on the same port.

Port sharing does not rely on an active sysplex distributor implementation; it works without a Sysplex distributor. Port sharing can be used in addition to sysplex distributor operations. As of this writing, z/OS supports two modes of port sharing: SHAREPORT and SHAREPORTWLM.

### SHAREPORT

Incoming client connections for a configured port and interface are distributed by the TCP/IP stack across the listeners that use a weighted round-robin distribution method that is based on the Server accept Efficiency Fractions (SEFs). The SEF is a measure of the efficiency of the server application (it is calculated at intervals of approximately one minute) in accepting new connection requests and managing its backlog queue, as shown in Figure 8-5.

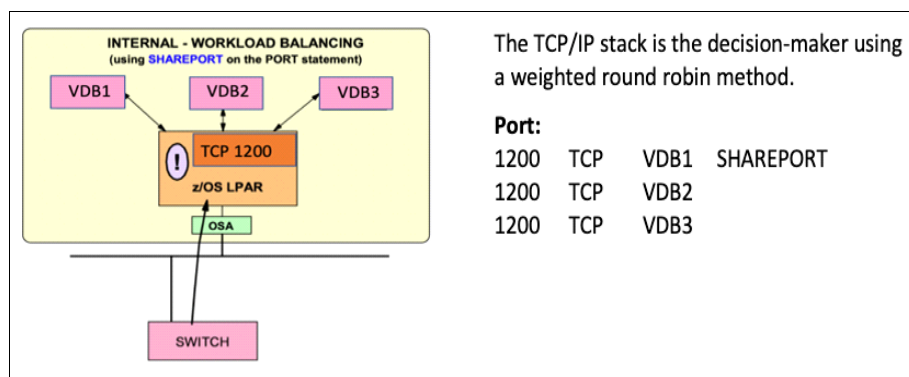


Figure 8-5 SHAREPORT configuration

## SHAREPORTWLM

Similar to SHAREPORT, SHAREPORTWLM causes incoming connections to be distributed among a set of TCP listeners. However, unlike SHAREPORT, the listener selection is based on WLM server-specific recommendations, and modified by the SEF values for each listener. These recommendations are acquired at intervals of approximately one minute from WLM, and they reflect the listener's capacity to handle more work (see Figure 8-6).

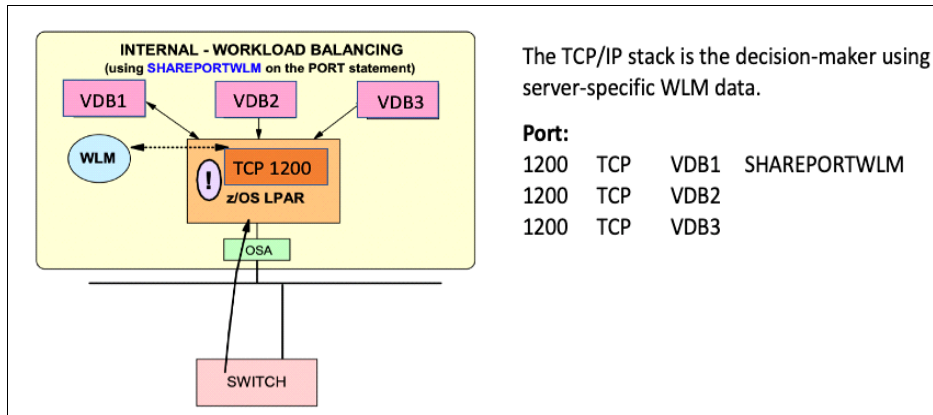


Figure 8-6 SHAREPORTWLM configuration

## WLMHEALTHREPORT, SHAREPORTWLM, and the DVM server

The DVM server reports a health status to WLM to allow WLM to better manage which DVM server to pass an inbound connection when SHAREPORTWLM is used.

The WLMHEALTHREPORT parameter must be set to YES, which is the default setting. An informational message similar to the following example is available with the server WLMHEALTHREPORT support to indicate when the health status of the DVM server changes:

```
VDB1DB0724I Server VDB1 WLM health changed from 100% to 90%.
```

Depending on the severity or frequency of errors, the health setting percentage can be reduced further and extra messages issued. After the health of the server increases, an extra message is generated similar to the following example:

```
VDB1DB0724I Server VDB1 WLM health changed from 90% to 100%.
```

The Server Trace Browse can be used to further investigate abrupt changes in WLM Health Status. No change is required for the DVM server configuration member AVZSIN00 to support SHAREPORT or SHAREPORTWLM.

## Sysplex load balancing with DVIPA

The distributor stack is a network-connected stack that owns a specific VIPA address and acts as the distributor for connection requests. The target stack is the owner of DVM server instances, to which the distributing stack forwards the requests. Together, they are called participating stacks for the sysplex distributor.

All participating z/OS images communicate through XCF, which permits each TCP/IP stack to have full knowledge of IP addresses and server availability in all stacks. Sysplex distributor for z/OS-integrated intra sysplex workload balancing of DVM servers is shown in Figure 8-7.

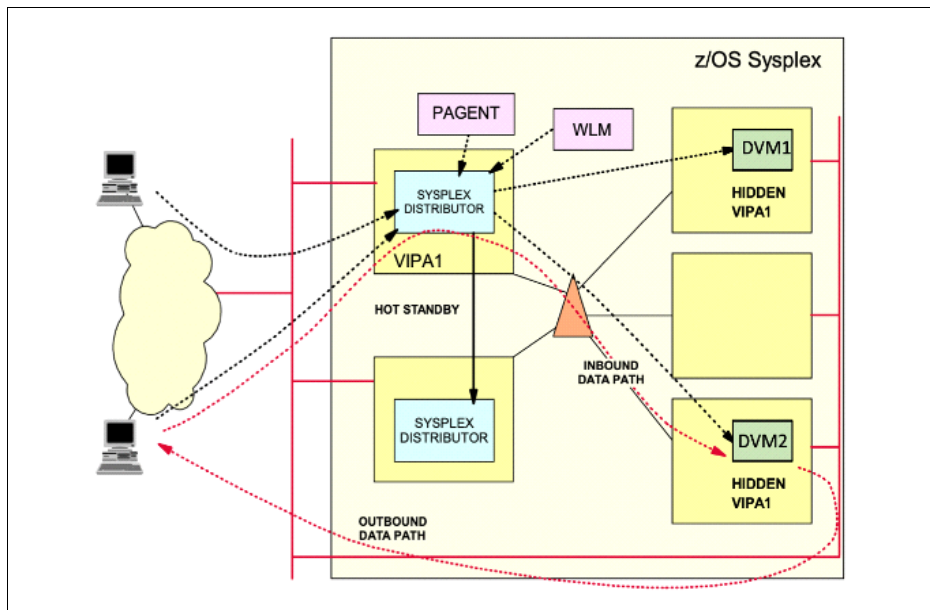


Figure 8-7 Sysplex load balancing using DVIPA

Sysplex distributor provides an advisory mechanism that checks the availability of DVM servers that are running on separate z/OS images in the same sysplex and then selects the best-suited target server for a new connection request. The Sysplex distributor bases its selections on real-time information from IBM Workload Manager (WLM). Sysplex distributor also measures the responsiveness of target servers in accepting new TCP connection setup requests, favoring those servers that are accepting new requests.

When the selection of the target stack is complete, the connection information is stored in the sysplex distributor stack to route future IP packets that belong to the same connection as the selected target stack. Routing is based on the connection (or session) to which IP packets belong, which is known as *connection-based routing*.

Configuring DVIPA and Sysplex Distributor are done within the TCP/IP stack and no components are in DVM server, which must be configured. After the Sysplex is configured to enable dynamic linking, any inbound connections can issue a CALL to WLM to check, which is the ideal stack to which the connection is routed.

Configure DVIPA by using IBM Documentation and define a VIPADYNAMIC section in the TCP/IP profile, as shown in the following example:

```
VIPADYNAMIC
 VIPADEFINE 255.255.255.0 10.17.100.60
 VIPADISTRIBUTE
 DISTMETHOD BASEWLM 10.17.100.60
 PORT 2200
 DESTIP
 192.168.1.1
 192.168.1.2
ENDVIPADYNAMIC
```

The DVIPA address is 10.17.100.60. The further definition (DISTMETHOD BASEWLM) states to perform WLM distribution of all inbound requests that are received by port 2200. Plan on having one DVIPA address and one non-DVIPA address for those applications and connections that do not need broadcast. To have the DVM server's TCP/IP listener listen on two IP addresses (one DVIPA and one non-DVIPA), the extra parameters in the following example must be set:

```
IF DoThis then
 "MODIFY PARM NAME(ALTERNATEIPADDRESS1) VALUE(10.17.100.60)"
 "MODIFY PARM NAME(DVIPABINDALL) VALUE(YES)"
END
```

It is recommended to use ALTERNATEIPADDRESS1 as the DDVIPA address. ALTERNATEIPADDRESS2 is another optional parameter that can be used to specify a third IP address.

### 8.5.3 Load balancing with CICS regions

The DVM server manages load balancing for CICS regions by using the LOADBALGROUP parameter in the CICS definition for the CICS server configuration member. Define the following statements in IN00, as shown in the following example:

```
"DEFINE CONNECTION NAME(AAAA)", "GROUP(AAAA)",
"ACCESSMETHOD(IRC)", "NETNAME(SDBAAAA)", "INSERVICE(YES)", "PROTOCOL(EXCI)",
"APPLID(CICSJ)", "LOADBALGROUP(LBG1)" "DEFINE CONNECTION NAME(BBBB)",
"GROUP(BBBB)", "ACCESSMETHOD(IRC)", "NETNAME(SDBBBBB)", "INSERVICE(YES)",
"PROTOCOL(EXCI)", "APPLID(CICSL)", "LOADBALGROUP(LBG1)"
```

These statements tell the DVM server the CICS region associated with the corresponding LOADBALGROUP (CICSJ or CICSL) and how to send the request. The DVM server often performs this operation in a round robin fashion.

If one CICS that belongs to the LOADBALGROUP becomes INACTIVE (for example, CICSJ), the DVM server sends a new CICS request to the other CICS (CICSL), which is part of the same LOADBALGROUP. Many CICS regions can exist for the same LOADBALGROUP.

### 8.5.4 Db2-Direct and IMS-Direct

Db2-Direct and IMS-Direct are features that are provided by the DVM server to directly access the back-end datasets of Db2 for z/OS and IMS by bypassing the respective database managers for improved performance and reduced elapsed time. This feature can be used for READ-ONLY operations, which do not require data integrity of the latest database UPDATE operations that are similar for analytical queries. Figure 8-8 on page 178 shows bypassing the underlying database subsystems for I/O.

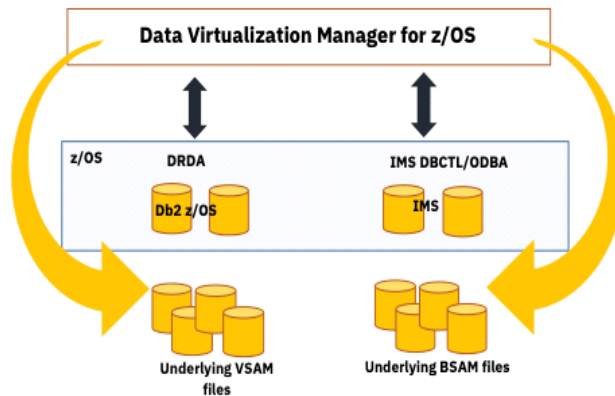


Figure 8-8 DVM server bypass of database I/O subsystems

## Db2-Direct

Db2-Direct is a DVM server access method that reads Db2 VSAM linear datasets directly, outside the Db2 address space, instead of accessing the data through traditional Db2 APIs. Large data pulls can be performed in service request block (SRB) mode with MapReduce and VPD features without any prerequisite processing, such as the collection of statistics that use the DVM command DRDARange. Db2-Direct allows READ-ONLY access to the data. It also provides a significant benefit in performance and reduced elapsed time in processing analytical queries.

Transactional integrity is not guaranteed because of the absence of record-level locking during reading activity. Security is managed by using Db2 table authorization.

Consider the following points about DB2-Direct:

- ▶ Db2-Direct does not support tables with edit procedures or SQL statements that contain joins, LOB columns, or key columns.
- ▶ The Db2 subsystem that is hosting a Db2 table must be active when Db2-Direct enabled tables are loaded or refreshed. The MAP building requires Db2 system access to identify data set information in the Db2 system catalog.
- ▶ The DVM server requires READ access to the Db2 VSAM linear datasets and that datasets are available during map load or refresh for the virtual table.
- ▶ Virtual tables that are enabled for Db2-Direct must include all the columns that are defined in the base Db2 table as the column information is not available while loading directly from DB2 VSAM linear datasets.
- ▶ If Db2-Direct table security is enabled, the Db2 subsystem must be available to check security at SQL query time.
- ▶ Users can check the DVM server trace logs for the following messages to confirm whether DB2-Direct is enabled or used:
  - Startup: DB2 direct processing enabled for <map-name>
  - Running: Processing table that uses DB2 direct
  - Failure: DB2 direct processing disabled for <map-name>

By default, Db2-Direct is enabled in the DVM server. To disable the Db2-Direct feature for a virtual table, set the variable OPTBDIDD to 1 in a VTB rule. Db2-Direct can be disabled by using the following parameter in the DVM configuration file hlq.SAVZEXEC(AVZSIN00):

Disable: “MODIFY PARM NAME(DISABLEDB2DIRECT) VALUE(YES)”

For more information about DB2-Direct configuration, see [this web page](#).

## IMS-Direct

The IMS-Direct feature provides MapReduce and parallelism support for accessing native IMS files. This support bypasses the requirement of having to use native IMS API calls by reading the IMS database files directly, which is similar to how an unload utility can work. This method provides a significant performance improvement and reduces elapsed time in processing analytical queries.

The DVM server determines the best method of access to underlying IMS data. The DVM server chooses to activate IMS-Direct (native file support) or use IMS APIs. This determination is based on the database and file types that are supported and the size of the database. Virtual tables of IMS segments are required. IMS-Direct is supported by the following types of IMS Databases:

- ▶ Hierarchical direct access method (HDAM): VSAM and OSAM
- ▶ Hierarchical indexed direct access method (HIDAM): VSAM and OSAM
- ▶ Partitioned HDAM (PHDAM): VSAM and OSAM
- ▶ Partitioned HIDAM (PHIDAM): VSAM and OSAM
- ▶ Fast Path data entry database (DEDB)

Security is managed on the IMS native data set when IMS-Direct is used. The USERID of the client connection must have the necessary security permissions for reading the IMS database datasets. Transactional integrity is not guaranteed because of the absence of record-level locking during reading activity.

IMS-Direct can be enabled by changing the syntax of DontDoThis to if DoThis, and then, setting the parameter IMSDIRECTENABLED to YES. The following parameters in the DVM configuration file hlq.SAVZEXEC(AVZSIN00) are used to enable IMS Direct:

```
IF DoThis then do
 "MODIFY PARM NAME(IMSDIRECTENABLED) VALUE(YES)"
 "MODIFY PARM NAME(IMSDIRECTBUFFERSIZE) VALUE(1024)"
 "MODIFY PARM NAME(ACIINTSEGMP256) VALUE(200)"
 "MODIFY PARM NAME(TRACEIMSDBREFRESH) VALUE(YES)"
 "MODIFY PARM NAME(TRACEIMSDIRSTATS) VALUE(YES)"
 "DEFINE IMSDBINFO",
 . . .
END
```

## Enabling IMS-Direct

MapReduce must be enabled in the IN00 configuration file when IMS-Direct is turned on. If you are performing an INSERT, UPDATE, or DELETE, IMS-Direct switches to DBCTL automatically, even when IMS-Direct is enabled. Users can enable the trace option in the IN00 configuration file with the following parameters to confirm IMS-Direct is used while querying the table:

```
"MODIFY PARM NAME(TRACEIMSDIRTASKSETUP) VALUE(YES)"
```

For more information about IMS-Direct configuration, see [this web page](#).

## 8.5.5 Java Database Connectivity performance

Modern applications can connect to the DVM server that uses Java Database Connectivity (JDBC). This standard Java API is used for database-independent connectivity between the Java programming language and a wide range of databases. Specific properties are available to help influence performance characteristics. Data buffering, parallelism, and MapReduce are specific DVM server functions that can be dictated as part of the JDBC connection string that is used by the client application to significantly improve overall performance.

### JDBC and buffering data

When sending large amounts of data to the client or the server, one way to optimize performance is by choosing the suitable size of the buffer that the driver uses to send data. The JDBC driver communicates with the DVM server by using the Communication Buffer (CMBU) protocol, which specifies the number of buffers to be used. The maximum size of a buffer is set by using the `MaximumBufferSize` (MXBU) JDBC property and the `NETWORKBUFFERSIZE` parameter in the server configuration file.

The application can use the `MaximumBufferSize` JDBC property to optimize the query performance. The buffer size value that the JDBC driver uses is the result of a handshake between the driver and server.

When the driver logs on to the server, it requests for the `MaximumBufferSize`. If the `MaximumBufferSize` is greater than `NETWORKBUFFERSIZE`, the server tells the driver to use the `NETWORKBUFFERSIZE` in the log-on response. When setting the `MaximumBufferSize` value, consider the distance between the client and the server because network latency can harm performance.

The buffer size that works best for clients and servers that are closer in proximity (low latency) need not be the buffer size that works best when clients and servers are farther away (high latency). Figure 8-9 shows how an `INSERT` statement is sent by using multiple buffers to the DVM server.

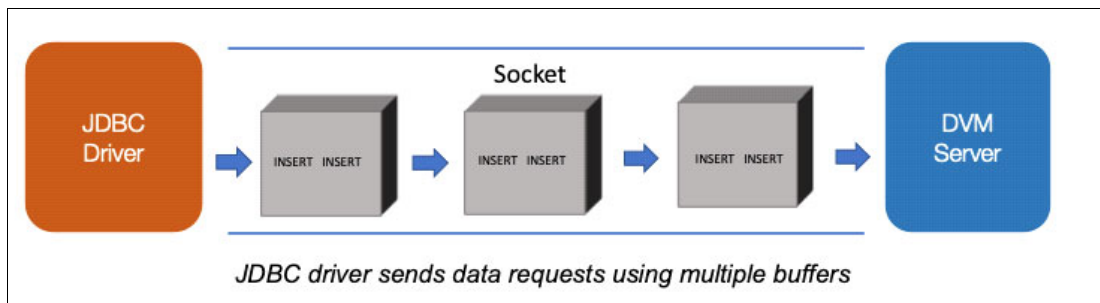


Figure 8-9 JDBC driver sending multiple buffers for `INSERT`

When you run a large SQL `INSERT` statement or a batch of `INSERT` statements, the statements are divided into multiple data buffers that are no larger than the size you specify for `MaximumBufferSize`. The buffers are then sent to the server.

When you run a large SQL `INSERT` statement or a batch of `INSERT` statements, the statements are divided into multiple data buffers of buffer size that is negotiated with the DVM server. The buffers are then sent to the server.



Figure 8-10 shows the server sending the client rows of data in a single buffer.

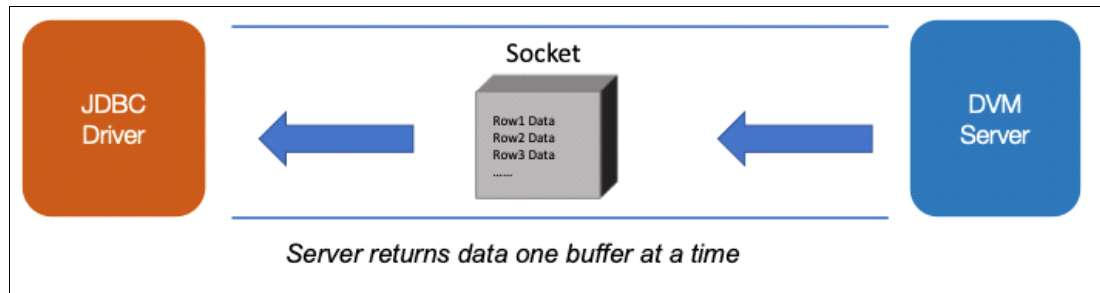


Figure 8-10 DVM server returns one buffer at a time

After running an SQL SELECT statement, the server divides and returns the requested rows of data in buffers that was sized, one buffer at a time. The client can call `next()` on the result set until all rows are read. When the `next()` call no longer has a row to read from the buffer, the driver issues a request to the server for another row buffer. This cycle continues until the client reads all rows from all buffers.

The following code provides a sample implementation:

```
Connection con = DriverManager.getConnection
("jdbc:rs:dv://host:port;Key=Value;Key=Value;",username,password);
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next())
{
 int x = rs.getInt("a");
 String s = rs.getString("b");
 float f = rs.getFloat("c");
}
```

The client can experience a pause during the `next()` API calls when all rows in the current buffer are read and the driver fetches another buffer of data. This pause can be avoided by enabling the parallel I/O.

### JDBC and parallel I/O

When parallel I/O is enabled, the JDBC driver creates multiple and separate threads by using DVM server buffers to pre-fetch, read, and insert rows into a queue. The JDBC driver works to ensure that this queue is fully used so at least one buffer always is ready for the `next()` API call to use. Pauses are eliminated so the client can continue to call the `next()` buffer in the results set. The process iterates until all rows in all buffers are read and queued.

Figure 8-11 shows row data being sent to the client by using the main and parallel thread. The ParallelIoBufferCount (PIOBC) property determines the number of buffers in use.

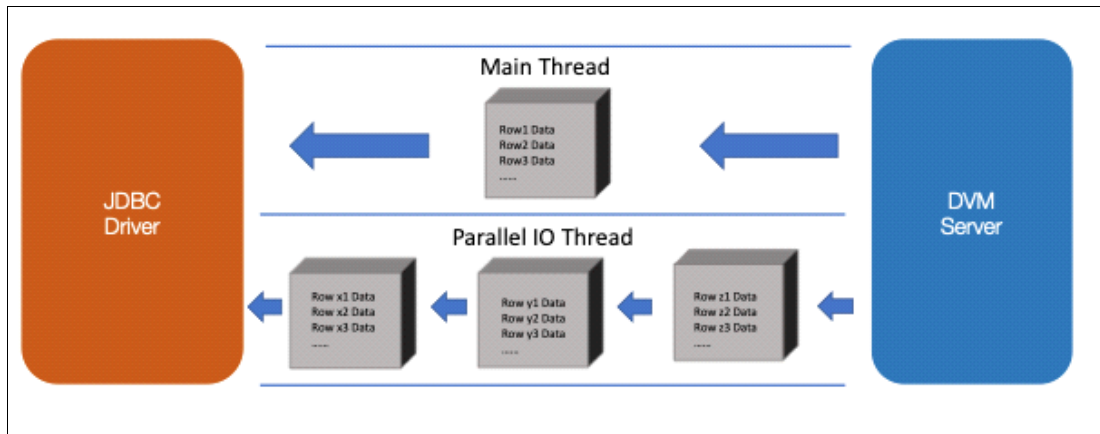


Figure 8-11 Parallel IO pre-fetches buffers

## JDBC and MapReduce

MapReduce is a JDBC driver-controlled feature that can be used to improve SQL query performance by reading results with multiple connections between the driver and server. The following types of MapReduce are available:

- ▶ Server-controlled MapReduce: MapReduce is performed on the server by using the JDBC driver.
- ▶ Client-controlled MapReduce: MapReduce is performed on the client and the JDBC driver manages single connections, whereas the client manages other connections and aggregate the results from each connection.

If you use MapReduce with RDBMS or IMS, you must complete metadata repository configuration requirements.

### **Server-controlled MapReduce**

The data mapping step creates threads that run a query by using more than one DVM server connection. Each thread maps one connection to one server. The reduce step reads results on each thread in parallel for each connection, and then transparently presents the data to the client, as shown in Figure 8-12 on page 183.

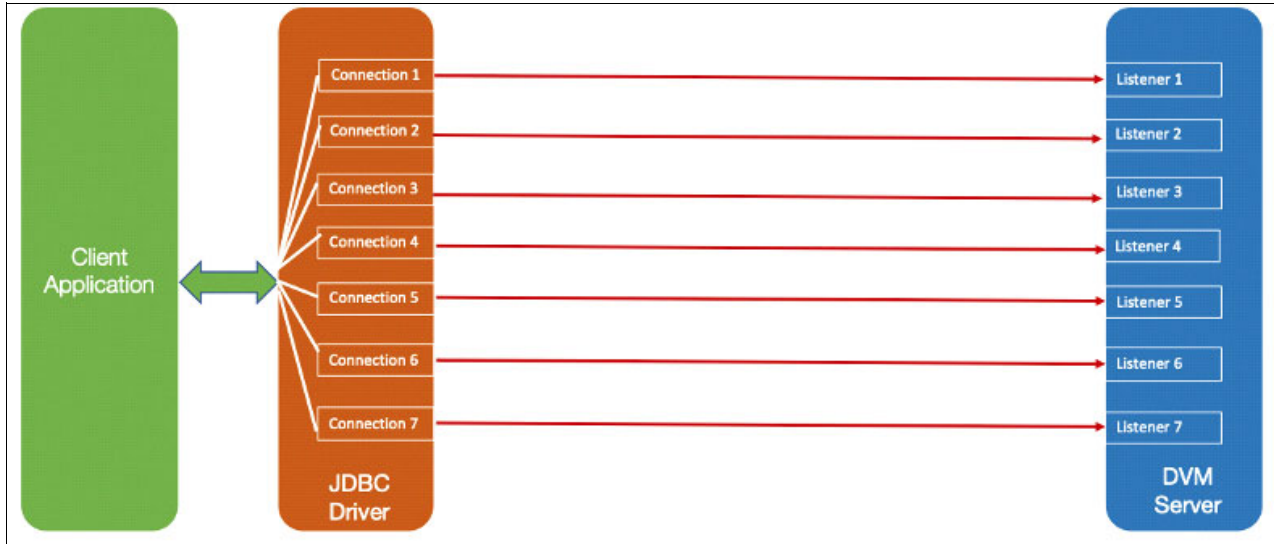


Figure 8-12 Server controlled MapReduce with a single DVM server

A single-server connection is configured by setting the MapReduceClient JDBC property:

```
MapReduceClient = (Hostname, Port, TaskCount)
MapReduceClient= (dvs.example.com, 9999, 7)
```

MapReduce also can be configured to use multiple server connections. Figure 8-13 shows a client that is connected to two servers with MapReduce.

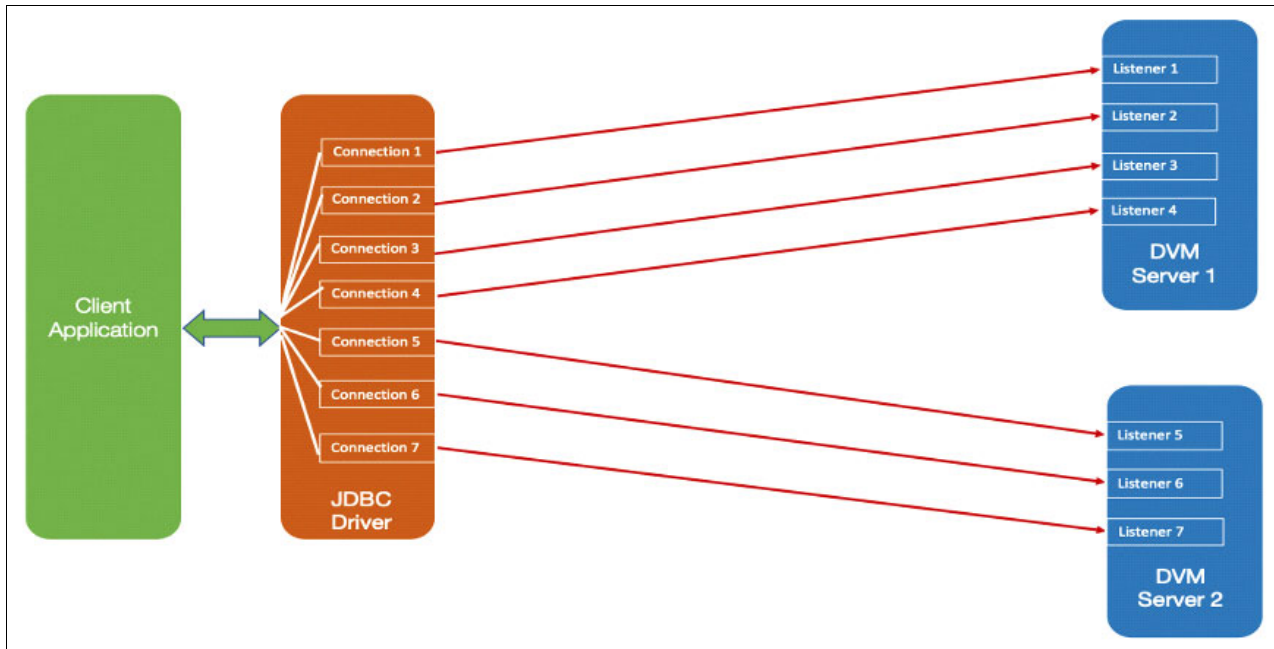


Figure 8-13 Server-controlled MapReduce with two DVM servers

Configure multiple server connections by setting the MapReduceClient property:

```
MapReduceClient = (Hostname1, Port, TaskCount1), (Hostname2, Port, TaskCount2)
MapReduceClient = (dvs1.example.com, 9999, 4), (dvs2.example.com, 1111, 3)
```

MapReduce also can be controlled to use a specific range of clients:

Example (clients 1 - 4 and 5 - 7)

```
MapReduceClient = (Hostname, Port, maxClientNo, startClientNo, endClientNo)
```

```
MapReduceClient = (host1, 1200, 7, 1, 4), (host2, 1200, 7, 5, 7)
```

### Client-controlled MapReduce

In some use cases, the client application can perform the MapReduce process by using Apache Spark and still benefit from the server's MapReduce feature. The JDBC driver supports this use case by specifying a single connection as the JDBC MapReduce connection. This connection is then available for use by a group of specified connections. The JDBC driver manages single connections. The client application must aggregate the results from each connection managed (see Figure 8-14).

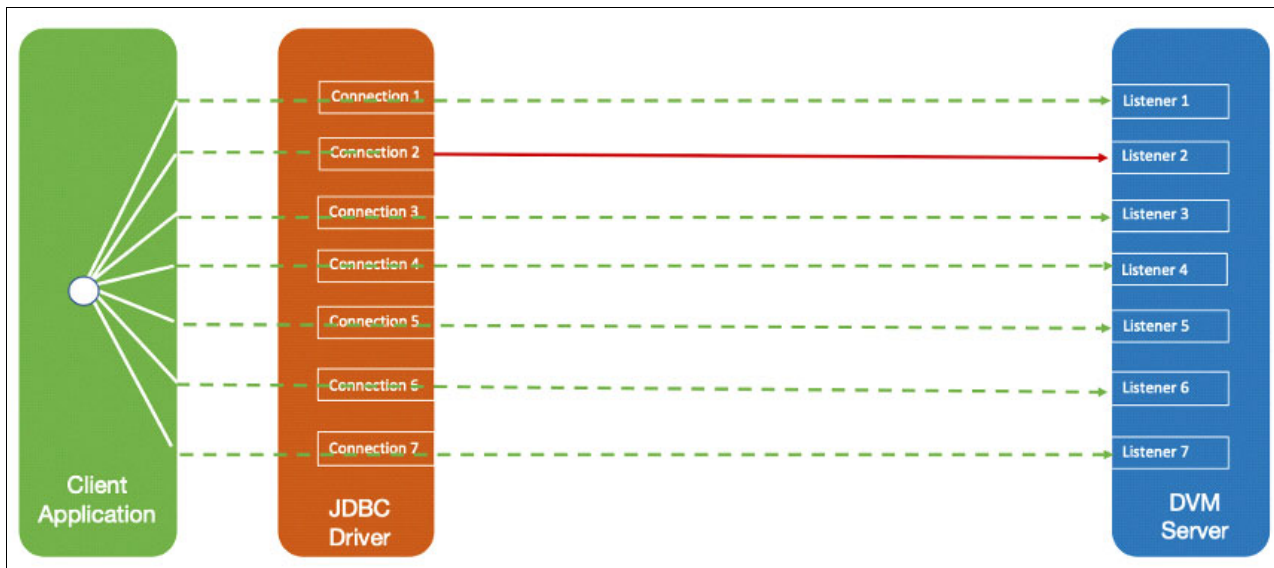


Figure 8-14 Client controlled MapReduce with seven parallel connections

The JDBC driver creates a single connection, which is indicated as connection 2 out of 7 available connections. By using a framework, such as Apache Spark, you must create and manage all remaining connections and aggregate the results from each of those connections.

Client-controlled MapReduce can be enabled by setting the JDBC driver properties `MapReduceClientCount` and `MapReduceClientNumber`.

Consider the following points:

- ▶ The `MapReduceClientCount` property is used to specify the total number of connections that are associated with the group of client connections.
- ▶ The `MapReduceClientNumber` property defines a specific client connection within the group is managed by the JDBC driver and has a value between 1 and the number that is specified for the `MapReduceClientCount` property.
- ▶ The JDBC driver runs queries by using the single MapReduce connection for the client connection that is specified in the `MapReduceClientNumber` property.
- ▶ Data for the specified connection are returned, as opposed to the use of `MapReduceClientCount` over one or more connections to get all rows of data.

- ▶ To configure client-side MapReduce, set the JDBC driver `MapReduceClientNumber` and `MapReduceClientCount` parameters, as shown in the following example:

```
MapReduceClientNumber, MapReduceClientCount
MapReduceClientNumber=2; MapReduceClientCount=7;
```

## 8.6 ODBC performance

The ODBC drivers are used by non-Java applications and tools to access data that is made available through the DVM server. The ODBC driver implements the ODBC Direct network protocol that is used to connect to the DVM server and uses the ODBC API to run SQL queries.

The DVM server ODBC driver for the Windows platform can be downloaded from the IBM Support Fix Central [download site](#).

The performance of ODBC drivers with DVM can be optimized by using connection pooling and optimized fetch. The connection pooling in the Windows platform can be configured through the ODBC Data Source Administrator. For UNIX and Linux platforms, connection pooling is managed by the ODBC Driver Manager.

When optimized fetch is enabled, rows ahead of the current row are asynchronously extracted before the client application requests them. This data is then returned to the client application in blocks that can be as large as 32 KB. Enabling optimized fetch helps to minimize network traffic and speeds subsequent fetches because the requested data is likely already in a returned block.

Optimized fetch is enabled by including the `R0=YES` connection property in a connection string (a connection string can be used with a DSN) or by appending the `FOR FETCH ONLY` clause to a `SELECT` statement.

The performance of ODBC connected applications also can be improved by using catalog functions, retrieving the required data, selecting functions that optimize performance, and managing connections efficiently. For example, catalog functions often are expensive, and minimize the use of catalog functions. Avoiding search patterns in catalog functions can improve performance. Similarly, the use of bound columns (for example, `SQLBindCol` instead of `SQLGetData`) and the use of `SQLExtendedFetch` instead of `SQLFetch` can help to improve the performance

## 8.7 Integrated Data Facility and DS Client API

DVM for z/OS provides an interface to access virtualized data from within more traditional mainframe languages, such as COBOL, Natural, or PL/I. With Db2 Integrated DRDA Facility (IDF), the traditional mainframe applications can use standard EXEC SQL statements to access established data sources, such as VSAM, IMS, and ADABAS.

The DS Client high-level API allows an application that is running on z/OS to use a call-level interface to communicate with DVM to process SQL requests and to retrieve result sets. The performance of Db2 IDF and DS Client APIs is similar for single user address space applications. However, Db2 IDF performed much better than DS Client API for multiuser environments, such as CICS and IMS.

Also, the applications can use the popular API with Db2 IDF compared to specific high-level API with DS Client. Therefore, it is recommended to use Db2 IDF to access data sources from traditional mainframe languages when Db2 is available in the customer environment. For those customer environments where Db2 is not available, DS Client APIs can be used.

The performance of DS Client API can be optimized with data buffering and MapReduce features that are similar to JDBC. The number of active client interface servers to process client requests also can be optimized by tuning the parameter ACIDVCLIENTMIN. The maximum number of MapReduce tasks that starts to process a request from a DS Client application can be specified by using ACIMAPREDUCECLIMAX.

Similarly, the default size (DSCLIENTBUFFERSIZ) of the buffer, the maximum (DSCLIENTBUFFERSIZEMAX) and minimum (DSCLIENTBUFFERSIZEMIN) buffer size, and the maximum number of buffers (DSCLIENTBUFFERNUMMAX) also can be specified.

## 8.8 Query optimization and performance

DVM for z/OS provides default values that are part of the standard installation in the IN00 configuration file that serves as standard level optimization. These defaults are fine-tuned over years of customer experience.

### 8.8.1 SQL best practices

Though increasing resources is an easy option, it is worthwhile to invest time into writing efficient SQL statements for the DVM server to use. The DVM server is an SQL engine that works with data sources that do not conform to relational database rules. A best practice is to conform to the most compliant SQL that matches the source data system, including SQL, functions, routines, and data type.

Consider the following points:

- ▶ Select only the required fields
- ▶ Understand how a Large and Small table is being JOINED
- ▶ Avoid scalar, arithmetic process into the SQLs

Consider the following best practices:

- ▶ Use simple predicate rewrites. Doing so causes the optimizer to generate significantly different query plans and force a different access path, which result in getting the best out of the SQL.
- ▶ Avoid table space scans when your goal is to fetch a much smaller subset of data.
- ▶ Sort only on the columns needed.
- ▶ Minimize the number of times cursors are opened and closed.

Designing SQL is an art form. Although many different statements result in the same output, variances exist in the execution time that are based on how the query is constructed. Table 8-4 provides some guidance for creating queries.

Table 8-4 Best practices for query design

| Category             | Description                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Efficient SQL        | Do not code mathematics on columns in predicates.                                                                                                                                                                                                              |
|                      | Sort only on the columns that are needed. No need to ORDER BY EMPNO, LASTNAME when you can ORDER BY EMPNO.                                                                                                                                                     |
|                      | Watch out for the LIKE predicate. Begins With logic is indexable. Contains is not indexable. Ends With is not indexable.                                                                                                                                       |
|                      | Do not code Not Between. Rewrite it as >HV or <HV.                                                                                                                                                                                                             |
|                      | Use Fetch First XX Rows whenever possible.                                                                                                                                                                                                                     |
|                      | Make sure cardinality statistics exist for all columns in all tables.                                                                                                                                                                                          |
|                      | Code Not Exists over Not In. Both are stage 2 predicates but Not Exists typically outperforms the Not In, especially if the list is long.                                                                                                                      |
|                      | When joining two tables the execution is faster if the larger table is on the left side of the join.                                                                                                                                                           |
|                      | Code WHERE clauses with columns that have unique or good indexes.                                                                                                                                                                                              |
|                      | Prioritize WHERE clauses to maximize their effectiveness. First code the WHERE column clauses that reference indexed keys, then the WHERE column clauses that limit the most data, and then the WHERE clauses on all columns that can filter the data further. |
| Good coding practice | When looking for a small set of records, try to avoid reading the full table by using an index and by providing any possible key values. You can also use more WHERE clauses so that the fetch goes directly to the actual records.                            |
|                      | All Case logic should have an else coded, which eliminates DB2 returning nulls by default if all the Case conditions are not met.                                                                                                                              |
|                      | Stay away from Not logic if possible.                                                                                                                                                                                                                          |
|                      | Minimize the number of times cursors are opened and closed.                                                                                                                                                                                                    |
|                      | Code stage 1 predicates only. Rewrite any stage 2 predicates.                                                                                                                                                                                                  |
|                      | Use FOR FETCH ONLY on all read only cursors.                                                                                                                                                                                                                   |
|                      | Reduce the number of rows to process early by using Sub-selects and WHERE predicates.                                                                                                                                                                          |
|                      | Avoid joining two types of columns and lengths when joining two columns of different data types or lengths. One of the columns must be converted to either the type or the length of the other column.                                                         |
|                      | Limit the use of functions against large amounts of data.                                                                                                                                                                                                      |

| Category                         | Description                                                                                                                                                                                                                                                                                                                     |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reduce impacts to the DVM server | Do not code functions on columns in predicates.                                                                                                                                                                                                                                                                                 |
|                                  | Minimize the number of times DB2 SQL statements are sent.                                                                                                                                                                                                                                                                       |
|                                  | Only select the columns that are needed.                                                                                                                                                                                                                                                                                        |
| Virtualization                   | Instead of using multi-level views, try to optimize your SQL queries. Creating views that call other views that call other views can result in joining to the same table multiple times when you only need it once. It creates millions of records in an underlying view where you are interested only in a handful of records. |

### Example of a non-optimized query

Consider the query in Example 8-1, which can take more than 30 minutes to run in a test environment.

*Example 8-1 Sample query before best practices*

---

Level 5 =====

```
SELECT SR_SYS_ID, ACCT_UNQ_ID,
ACCT_NUM, FULL_NAME, FIRST_NAME, MIDDLE_NAME,
LAST_NAME,
TAB1_ADDR_1, TAB1_ADDR_2, CITY, TAB1_PROV,
TAB1_POST_CD,
TAB1_CNTRY, S_IND_FLG,
INT_DEP_FLG,
INT_DEP, BNF_ID,
T_ACCT_TYP_CD, L_ENT_CD
FROM VIEW4;
```

Level 4 =====

#### **VIEW4**

```
SELECT A.SR_SYS_ID, A.ACCT_UNQ_ID, A.ACCT_NUM,
CAST(B.TAB2_NAME AS CHAR(255)) FULL_NAME, CAST(B.TAB2_FNAME AS CHAR(25))
FIRST_NAME,
CAST(B.TAB2_MNAME AS CHAR(10)) MIDDLE_NAME, CAST(B.TAB2_LNAME AS CHAR(25))
LAST_NAME,
B.TAB1_ADDR_1, TAB1_ADDR_2, CAST(B.TAB1_CITY AS CHAR(35)) CITY, B.TAB1_PROV,
B.TAB1_POST_CD, B.TAB1_CNTRY, A.S_IND_FLG,
A.INT_DEP_FLG, A.INT_DEP, A.BNF_ID,
A.T_ACCT_TYP_CD, A.L_ENT_CD
FROM VIEW_3A A
INNER JOIN VIEW_3B B ON A.ACCT_UNQ_ID = B.INP_ACCT_UNQ_ID
```

Level 3 =====

#### **VIEW\_3A**

```
SELECT SR_SYS_ID, ACCT_UNQ_ID, ACCT_NUM,
BNF_NAME, BNF_FNAME, BNF_MNAME,
BNF_LNAME, BNF_ADD1, BNF_ADD2, BNF_CITY,
BNF_PROV, BNF_PST_CD, BNF_CNTRY,
S_IND_FLG, INT_DEP_FLG, INT_DEP,
BNF_ID, T_ACCT_TYP_CD, L_ENT_CD
```



```
FROM VIEW_2A WHERE T_ACCT_TY_CD= '00999'
```

#### **VIEW\_3B**

```
SELECT A.INP_ACCT_UNQ_ID, A.INP_DEP_UNQ_ID, B.TAB2_NAME,
A.TAB2_FNAME, A.TAB2_MNAME, A.TAB2_LNAME, A.TAB1_ADDR_1,
A.TAB1_ADDR_2, A.TAB1_CITY, A.TAB1_PROV, A.TAB1_POST_CD, A.TAB1_CNTRY
FROM VIEW_2B A
INNER JOIN VIEW_2C B ON A.INP_ACCT_UNQ_ID = B.INP_ACCT_UNQ_ID
```

Level 2 =====

#### **VIEW\_2A**

```
SELECT A.SR_SYS_ID, A.ACCT_UNQ_ID, A.ACCT_NUM,
A.BNF_NAME, A.BNF_FNAME, A.BNF_MNAME,
A.BNF_LNAME, A.BNF_ADD1, A.BNF_ADD2, A.BNF_CITY,
A.BNF_PROV, A.BNF_PST_CD, A.BNF_CNTRY,
A.S_IND_FLG, A.INT_DEP_FLG, A.INT_DEP,
A.BNF_ID, A.T_ACCT_TYP_CD, A.L_ENT_CD
FROM TABLE4 A
INNER JOIN TABLE3 B ON A.ACCT_UNQ_ID = B.W4_ACCOUNT_UNIQUE_ID
```

#### **VIEW\_2B**

```
SELECT A.INP_ACCT_UNQ_ID, A.INP_DEP_UNQ_ID, B.TAB2_NAME,
B.TAB2_FNAME, B.TAB2_MNAME, B.TAB2_LNAME, C.TAB1_ADDR_1,
C.TAB1_ADDR_2, C.TAB1_CITY, C.TAB1_PROV, C.TAB1_POST_CD, C.TAB1_CNTRY
FROM TABLE1 A
INNER JOIN TABLE2 B ON A.INP_DEP_UNQ_ID = B.TAB2_DEP_UNQ_ID INNER JOIN TABLE5 C ON
A.INP_DEP_UNQ_ID = C.TAB1_DEP_UNQ_ID WHERE C.TAB1_PR_ADDR_FLG = 'Y' AND
UPPER(A.INP_PR_ADDR_H_FLG) = 'Y'
```

#### **VIEW\_2C**

```
SELECT INP_ACCT_UNQ_ID, GROUP_CONCAT(TAB2_NAME, ' AND ' , 255) AS TAB2_NAME
FROM VIEW1 GROUP BY INP_ACCT_UNQ_ID
```

Level 1 =====

#### **VIEW1**

```
SELECT A.INP_ACCT_UNQ_ID, A.INP_DEP_UNQ_ID, B.TAB2_NAME
FROM TABLE1 A INNER JOIN TABLE2 B ON A.INP_DEP_UNQ_ID = B.TAB2_DEP_UNQ_ID
```

---

From this query, we observe the following results:

- ▶ Each level of view is an SQL statement that must be parsed and validated.
- ▶ View1 causes a table scan for table1 and table2 and produces resultset1.
- ▶ View2a causes table scan on table4 and table3 and produces resultset2.
- ▶ View2b causes table scan of table1, table2, and table5, and produces resultset3.
- ▶ View2c reads resultset1 and creates resultset4.
- ▶ View3a reads resultset2 and creates resultset5.
- ▶ View3b reads resultset3 and resultset4 and creates resultset6.
- ▶ View4 reads resultset5 and resultset6 and creates resultset7.
- ▶ Client query reads resultset7 to create the final result set.

In conclusion, this non-optimized query features the following characteristics.

- ▶ The tables table1 and table2 are scanned twice.
- ▶ Seven intermediate result sets are created.

- Seven SQL statements are parsed and validated.

### An optimized version of the query

The query that is shown in Example 8-2 is a result of following good practices and optimizing the sample query. It takes less than 3 minutes to offer the same result.

#### Example 8-2 Optimized query

---

```

SELECT SR_SYS_ID, ACCT_UNQ_ID,
 ACCT_NUM,
 CAST(T5.TAB2_NAME AS CHAR(255)) AS FULL_NAME,
 CAST(TAB2_FNAME AS CHAR(25)) AS FIRST_NAME,
 CAST(TAB2_MNAME AS CHAR(10)) AS MIDDLE_NAME,
 CAST(TAB2_LNAME AS CHAR(25)) AS LAST_NAME,
 TAB1_ADDR_1, TAB1_ADDR_2,
 CAST(TAB1_CITY AS CHAR(35)) AS CITY,
 TAB1_PROV,
 TAB1_POST_CD,
 TAB1_CNTRY, S_IND_FLG,
 INT_DEP_FLG,
 INT_DEP, BNF_ID,
 T_ACCT_TYP_CD, L_ENT_CD
FROM TABLE4 T1
INNER JOIN (A)
TABLE3 T2
ON T1.ACCT_UNQ_ID = T2.W4_ACCOUNT_UNIQUE_ID
INNER JOIN (B)
TABLE1 T3
ON T1.ACCT_UNQ_ID = T3.INP_ACCT_UNQ_ID
INNER JOIN (C)
TABLE2 T4
ON T3.INP_DEP_UNQ_ID = T4.TAB2_DEP_UNQ_ID
INNER JOIN (D)
 (SELECT INP_ACCT_UNQ_ID,
 GROUP_CONCAT(TAB2_NAME, ' AND ') AS TAB2_NAME
 FROM TABLE1 T5A
 INNER JOIN (resultset1 join)
 TABLE2 T5B
 ON T5A.INP_DEP_UNQ_ID =
 T5B.TAB2_DEP_UNQ_ID
 GROUP BY INP_ACCT_UNQ_ID) T5
ON T1.ACCT_UNQ_ID = T5.INP_ACCT_UNQ_ID
INNER JOIN (E)
TABLE5 T6
ON T3.INP_DEP_UNQ_ID = T6.TAB1_DEP_UNQ_ID
WHERE T_ACCT_TY_CD= 00999
AND T6.TAB1_PR_ADDR_FLG = 'Y'
AND T3.INP_PR_ADDR_H_FLG = 'Y

```

---

By combining View1 and View2C into one sub-select, we reduce overhead of parsing, validation, and result set caching:

View1

```
SELECT A.INP_ACCT_UNQ_ID, A.INP_DEP_UNQ_ID, B.TAB2_NAMEFROM TABLE1 A INNER JOIN
TABLE2 B ON A.INP_DEP_UNQ_ID = B.TAB2_DEP_UNQ_ID
```

This creates a result set to be used by View2C, including a column that is never used, A.INP\_DEP\_UNQ\_ID:

```
View2C
SELECT INP_ACCT_UNQ_ID, GROUP_CONCAT(TAB2_NAME, ' AND ', 255) AS TAB2_NAMEFROM
VIEW1 GROUP BY INP_ACCT_UNQ_ID
```

The consolidated select statement JOINS across View1 and View2C by using a unique common field. In this example, column T5A.INP\_DEP\_UNQ\_ID and T5B.TAB2\_DEP\_UNQ\_ID are the JOIN columns between the two views:

```
SELECT INP_ACCT_UNQ_ID,
GROUP_CONCAT(TAB2_NAME, ' AND ') AS TAB2_NAME
FROM TABLE1 T5A
INNER JOIN
TABLE2 T5B
ON T5A.INP_DEP_UNQ_ID =
T5B.TAB2_DEP_UNQ_ID
GROUP BY INP_ACCT_UNQ_ID
```

This creates resultset1, which is equivalent to the output of view2c that is used by JOIN (D):

- ▶ JOIN (A) uses T1 and T2 to produce derived table TA
- ▶ JOIN (B) uses TA and T3 and expands TA to TA+T3
- ▶ JOIN (C) uses TA+T3 and T4 and expands TA+T3 to TA+T3+T4
- ▶ JOIN (D) uses TA+T3+T4 and resultset1 and expands TA+T3+T4 to TA+T3+T4+resultset1
- ▶ JOIN (E) uses TA+T3+T4+resultset1 and T6 and expands TA+T3+T4+resultset1+T6
- ▶ The WHERE criteria is then applied to TA+T3+T4+resultset1+T6 to produce the final result.

In conclusion, this optimized query features the following characteristics:

- ▶ Each table is scanned only once.
- ▶ Only two SQL statements have to be parsed and validated.
- ▶ Only one derived table is created and augmented by each JOIN.
- ▶ Less data is cached.

## 8.8.2 Performance testing with Apache JMeter

The Apache JMeter application is open source software, a 100% pure Java application that load tests functional behavior and measures performance. Customers can use JMeter to measure the performance of the services and queries run on the DVM server.

Apache JMeter can be download from [this web site](#).

Complete the following steps to configure the DVM server for use with Apache JMeter:

1. Copy the DVM for z/OS client JDBC driver into the lib directory of the Apache JMeter installation directory. The JDBC driver for DVM is available with the driver installation member AVZBIN2.
2. Rename the file to JDBCdriver.zip and extract its content into the lib directory of the Apache JMeter installation directory.

After Apache JMeter is installed along with the DVM for z/OS JDBC driver, you can begin to define a thread group, which is a pool of users that run a specific test case per your SLA. You can consider a thread group as a Virtual User Group that perform a set of steps that you recorded. The Thread Group element helps to define a performance test scenario in JMeter.

For JMeter to connect to access mainframe data through the DVM server, a JDBC connection is required to ensure location and authorization credentials are defined for a successful connection. JMeter calls this a *database connection pool*.

After the thread group and connection pool are defined, JMeter uses the database connection pool to issue a named JDBC request or basically, an SQL statement to be run. Each JDBC request allows for several different JDBC configuration elements to be used, such as parameter values, parameter types, variable names, query timeout, and others.

Thread groups also can have a summary report generated for run tasks. The summary report creates a table row for each differently named request in your test and considers the total time over which the requests were generated. A sample summary report for JDBC request VSAM.STAFF is shown in Figure 8-15.

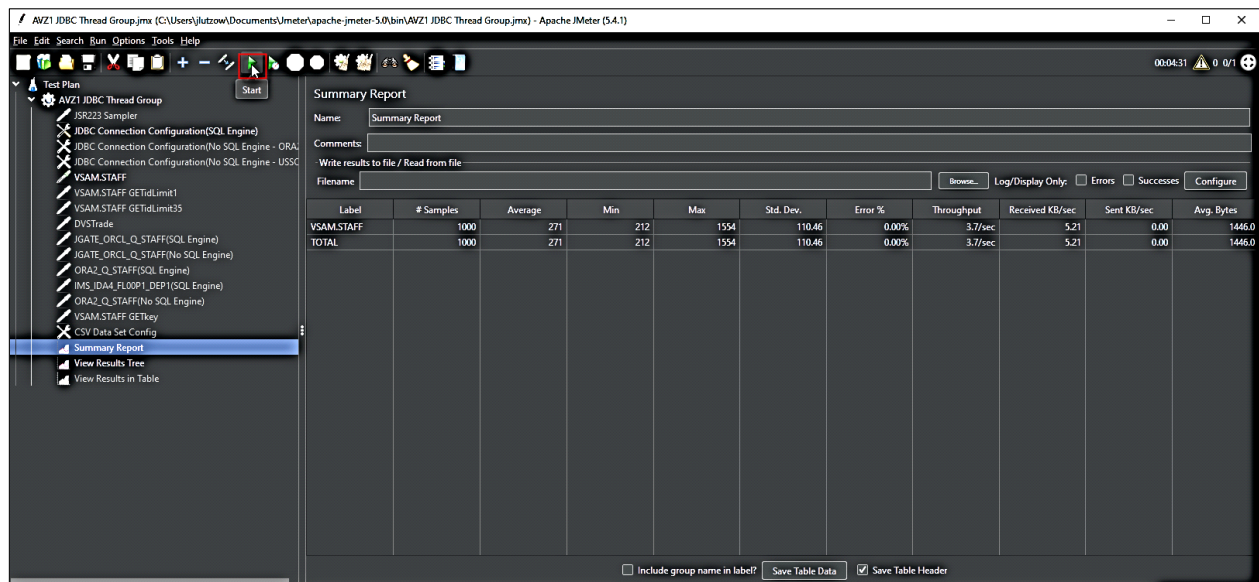


Figure 8-15 Apache JMeter

Based on the number of samples set, JMeter runs the JDBC query for the number of iterations that is defined and provides the average, minimum, and maximum elapsed time and throughput for the performance evaluation. The performance evaluation can be carried out further by testing with different query types and number of connections.

### 8.8.3 Performance testing by using the Command Line Tester

You can use the Command Line Tester (CLT) for evaluating the performance of specific queries, understand the number of rows, bytes read, elapsed time, and so on. CLT was developed by IBM specifically for DVM for z/OS and works to simplify development, troubleshooting, and diagnosing problem queries.

This tool also is used through RESTful API calls and includes the following features:

- ▶ No limit on the number of rows processed.
- ▶ Reports summarized execution time for data transfer.
- ▶ Isolates performance issues from application requesters.
- ▶ Uses ANSI SQL ? Flexible OUTPUT options.
- ▶ Command-line driven.
- ▶ Validated through a RESTful API.

The Command Line Tester utility is a preferred approach instead of DVM Studio for verifying query performance. DVM Studio is not meant to be used for performance testing. (As of this writing, it is not included in the DVM for z/OS packaged software or as part of PTF maintenance releases.)

CLT is not a host application that runs on the mainframe; rather, it is a Java-based application that runs from a command prompt on MS-Windows or a terminal on Linux (see Figure 8-16).



```
Command Prompt
C:\DVM\CLT>java -cp .* com.rs.jdbc.dv.CommandLineTester -url "jdbc:rs:dv://rs01:1211;dbty=DVS;user=*****;password=*****" --sql "SELECT * FROM STAFF4M" --verbose

SQL statement: SELECT * FROM STAFF4M
Total row count read: 4000000
Total bytes read: 112406372
Total bytes read (on wire): 112400130
Elapsed time: 84,058 milliseconds
```

Figure 8-16 CLT output reading a VSAM file with 4,000,000 records

It is encouraged to avoid the use of DVM Studio for query performance evaluation. DVM Studio includes limits for the number of rows that can be displayed, and the amount of HEAP space that is available.

## Setup

This package comprises the test harness jar files and includes log4j2. The easiest approach is to copy the JDBC driver jar (for example, dv-jdbc-3.1.xxx.jar) from your production area to the same directory as these tester jar files.

## Basic usage

Run the tool by using Java from a command prompt:

- ▶ Windows: `java -cp .* com.rs.jdbc.dv.CommandLineTester`
- ▶ UNIX/Linux: `java -cp .* com.rs.jdbc.dv.CommandLineTester <options>`

## Options

Use the `--help` option to display the usage options, as shown in the following example (run a query and display the elapsed time):

```
java -cp .* com.rs.jdbc.dv.CommandLineTester -help
java -cp .* com.rs.jdbc.dv.CommandLineTester --url "jdbc:rs:dv..." --sql "SELECT * FROM VT"
```

### **Multi-threaded MapReduce client test by using MRCC/MRCN**

Use the `CommandLineMrcnTester` class to run a multi-threaded MapReduce client test by using MRCC/MRCN:

```
java -cp .;* com.rs.jdbc.dv.CommandLineMrcnTester --url "jdbc:rs:dv://host:1234;MRCC=4; ..." --sql "SELECT ..." --verbose
```

The URL must include the `MapReduceClientCount/MRCC` property, and then the tool adds the suitable `MapReduceClientNumber/MRCN` setting for each separate connection. The `--verbose` flag displays the per-thread fetched row count and other information.



# Capacity planning and deployment

This chapter helps the user to understand basic principles that are needed in planning for an initial setup of the DVM for z/OS software and some estimation around capacity as workloads and data expand and grow in your environment.

The content in this chapter is not a comprehensive listing of areas to explore when assessing the size and capacity for a DVM server. Instead, it provides a generalized approach to capacity planning. Each customer environment is different and governed by different objectives, workload characteristics, and unique hardware and software needs.

This chapter includes the following topics:

- ▶ 9.1, “Capacity planning” on page 196
- ▶ 9.2, “Monitoring workloads” on page 199
- ▶ 9.3, “Capacity planning for future growth” on page 202
- ▶ 9.4, “Scaling for growth with the DVM server” on page 203
- ▶ 9.5, “Workload balancing” on page 206
- ▶ 9.7, “Best practices for deploying the DVM server” on page 207
- ▶ 9.8, “Developing queries” on page 208
- ▶ 9.9, “Administering the DVM server in production” on page 211

## 9.1 Capacity planning

The objective of capacity planning is to have enough spare capacity to handle any likely increases in workload, and enough buffer capacity to absorb normal workload spikes between planning iterations. Capacity planning helps organizations operate efficiently.

DVM for z/OS can service many virtual tables over 35 different data sources that are running natively on all versions of z/OS, including z15 for 32-bit and 64-bit instruction sets. The inherent architecture uses MapReduce's capability to create parallel threads at the DVM server level, and the client driver for inbound requests.

DVM for z/OS virtualization architecture is zIIP eligible and works to help balance processing capacity for specific applications without affecting their total million service units (MSU) rating or machine model designation. When workload is dispatched onto a zIIP processor, the processing cycles that are used do not contribute to the MSU count; therefore, they do not affect software usage charges. Adding applications to IBM Z is more cost-effective, especially when compared to competing platforms, such as distributed systems or public cloud.

IT capacity planning revolves around how an organization can meet the storage, computer hardware, software, and connection infrastructure demands over time. Organizations require flexibility in scaling resources (typically the servers, CPU, and memory).

The DVM for z/OS architecture provides key technologies to facilitate the needs of any organization. One or more DVM servers can be configured to address the following needs:

- ▶ Improved user concurrency.
- ▶ The ability to distribute workloads.
- ▶ High availability for:
  - When failover or relocation of workloads is required
  - Outages that are related to a hardware problem, network failure, or required maintenance
- ▶ zIIP eligibility; facilitates cost reduction for production environments and provides more CPU processing availability by delivering up to 99% utilization.
- ▶ MapReduce improves elapsed time reduction and overall performance.
- ▶ DVM server memory allows the execution of complex SQL with the benefits of improved response time. Server memory or cache maintains the metadata (as opposed to physical data) to enhance SQL processing.
- ▶ Key-based access methods provide direct access to back-end data files that are servicing popular databases that are running on the mainframe, such as Db2 for z/OS and IMS.

Key capacity planning areas are specific to the physical server, available storage (on disk and in memory), network, power, and cooling. Cost drives discussions for new implementations and projected demand and growth.

Capacity planning must accommodate a balance of workloads that are involved in the cloud, external service providers or hosted solutions, and an overall increase in the number of networked devices.

It is commonplace to run wide-ranging workloads (mixtures of batch, transactions, web services, database queries, and updates) that are driven by transaction managers, database systems, web servers, and message queuing and routing functions.



The following primary workload types are used to evaluate capacity planning. Each workload type has unique effects on resources with the added independent variable of time:

- ▶ Transaction-based workloads

Typically, small amounts of data that are processed and transferred per transaction that is delivered to many users. Interactions between the user and the system are nearly real time. Mission-critical applications require continuous availability, high performance, data protection, and data integrity. Transaction examples include ATM transactions, such as deposits, withdrawals, inquiries or transfers, supermarket payments with debit or credit cards, and online purchases.

- ▶ Web-based workloads

Enterprises shift functions and applications to online activity for greater access and improved usability through an application programming interface (API), such as Java, where they can be written to run under a Java virtual machine (JVM). The advantage is that applications become independent of the operating system in use.

- ▶ Batch processing

Run without user interaction, where the job reads and processes data in bulk, perhaps terabytes of data, and produces output, such as billing statements. Mainframe systems are equipped with sophisticated job scheduling software with which data center staff can submit, manage, and track the execution and out-of-batch jobs. Key characteristics are larger amounts of data that represent many records, run in a predefined batch window, and can consist of the execution of hundreds or thousands of jobs in a pre-established sequence.

- ▶ Mixed workloads

Consist of a combination of online transactions, web-based, and batch processing that perform analytic processing, real-time analytics, Machine Learning, and Artificial Intelligence.

As a result, organizations must monitor their current capacity to establish a baseline and understand future business needs for any capacity that is needed to scale as the organization grows. z/OS enables capacity provisioning, workload management, fine-tuning your system for prioritization, scheduling, and resource allocation through WLM, z/OSMF, RMF, and CPM.

As a mainframe middleware application, the DVM server also can fine-tune based on workload demands. The DVM server determines the optimal method to access the underlying data for the various workload types (see Figure 9-1).

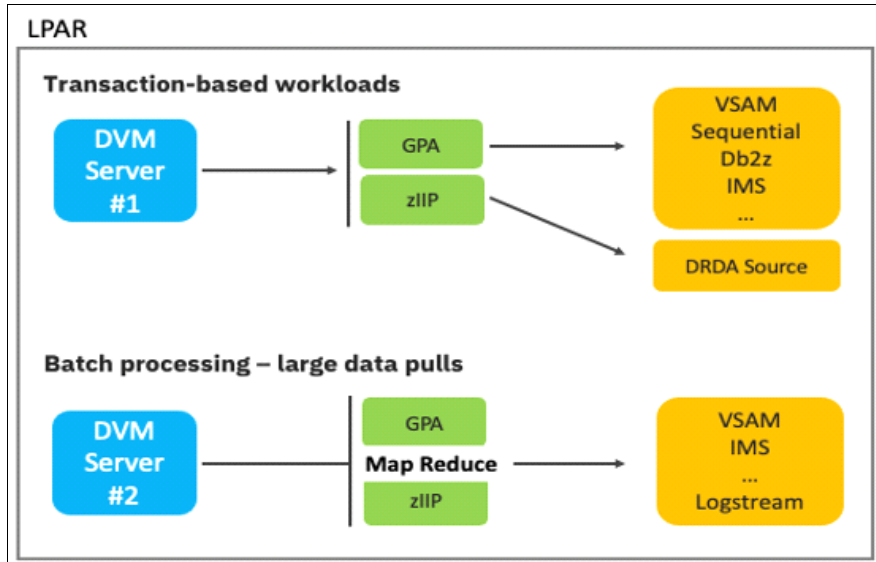


Figure 9-1 DVM server manages both transaction and batch processing

Table 9-1 lists models that are based on the current processing volume for any business and the maximum resources that are required for the next planning iteration.

Table 9-1 Initial recommendations based on workload type

| Workload type                       | Number of servers | GPA          | zIIP processors |
|-------------------------------------|-------------------|--------------|-----------------|
| Transactional                       | 1 - 5             | 16 Gigabytes | 4               |
| Batch processing (large data pulls) | More than 5       | 32 Gigabytes | More than 5     |

The volume of data within a data source and the type of Data Manipulation Language (DML) operations against that data are key considerations for assessing capacity planning. DML operations that represent complex JOINS with predicates, GROUP BY clauses, or aggregate functions require more processing to complete.

It is possible to configure multiple JDBC Gateway Servers along with the DVM server to access distributed (non-mainframe) data to accommodate high transactional processing or reading large volumes of data without negatively affecting latency or server processing. The amount of memory that is allocated to user space can be adjusted to better accommodate a larger number of users and overall concurrency.

## 9.2 Monitoring workloads

Workload management is critical in defining, assigning, reporting, monitoring, and managing performance goals. These activities help you understand how many resources, such as CPU and storage, can help meet demands for a specific environment.

IBM provides foundation tools for monitoring the end-to-end health and capacity of a system or LPAR. These tools are widely available and likely core to any mainframe environment:

- ▶ Tools provided by IBM for capacity planning (monitoring)
- ▶ System Management Facility (SMF)
- ▶ Resource Monitoring Facility (RMF)
- ▶ Capacity Provisioning Manager (CPM)

The DVM server can virtualize SMF records to help monitor the overall capacity of the DVM server and manipulate output through standard ANSI SQL syntax. The DVM server also features an ISPF window and workload generators to help evaluate the capacity and performance levels of the DVM server.

A command-line tester utility also can be used to help measure the performance of the DVM server. This utility can be requested from IBM Support and downloaded for your use. As of this writing, it is not included in the DVM for z/OS packaged software or as part of PTF maintenance releases.

### 9.2.1 Monitoring capacity with SMF records

The SMF data that is recorded as part of Record type 72 Sub Type 3 provides the resource consumption and response time of the DVM server.

Figure 9-2 shows key workload management elements within a service definition, including service policy, resource group, workload, service class, service class period number, and goals.

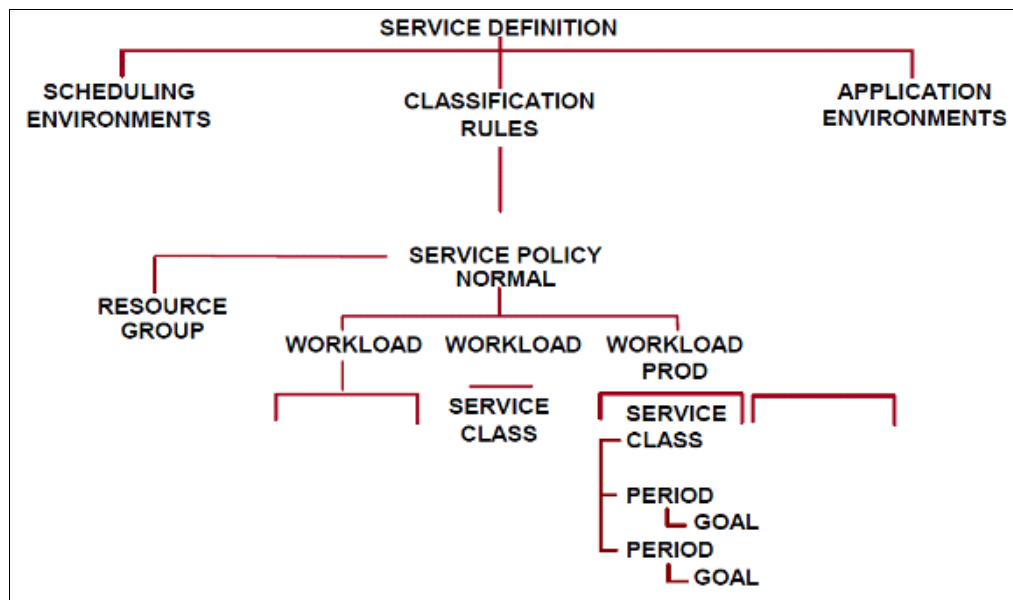


Figure 9-2 Workload management elements

The SFM 72 subsystem includes the following sections:

- ▶ The Workload Manager Control section provides SMF output to understand details that are related to the service policy, workload, service report, class name, and workload data.
- ▶ The Service Class Served section shows the name of the service classes that are served and the number of times the address space runs in the named service class.
- ▶ The Resource Group section contains information about the resource group to which the service class or tenant report class belongs. Each Resource Group has minimum and maximum capacity values for different resource types:
  - Percentage of the LPAR share
  - Percentage of a single processor capacity
  - Memory Limit
  - MSU/H
  - Combined general purpose and specialty processor consumption
- ▶ The Service/Report Class Period section contains goals and measured values for each service and reports class period.
- ▶ The Response Time Distribution section contains response time distribution buckets that are built around a midpoint that is calculated based on the goal of the associated service class period. The response time distribution count table shows the number of transactions that completed as a percentage of the midpoint.

SMF output from all these sections can be virtualized by creating virtual views that contain SQL queries. The SMF output helps analyze the capacity requirements in terms of CPU or Storage.

## 9.2.2 Monitoring performance by using SMF Record 249 Subtype 06

DVM SMF 249 Subtype 6 provides a more granular output when compared to a standard SMF 30 record. The DVM SMF 249 can state where the CPU is used more specifically and can log each inbound client request in the DVM server.

Each SMF record type 249 record contains information about all the work that was performed on behalf of a requesting client for each transaction. Inbound client requests can trigger zero, one, or more SQL operations to be run. Many Subtype 06 SMF records can be written in high-volume environments because a single SMF record is created for each transaction:

- ▶ Detailed CPU time based on Server (client/server request)
- ▶ Detailed CPU time based on Server (non-SOAP web request)
- ▶ Detailed output on each SQL statement run
- ▶ CPU and zIIP details based on each query by combining data from Type 01 and Type 0

AVZ.STORAGE is used to record the private and virtual storage that is used by the DVM server address space in 15-minute intervals. Table 9-2 lists the elements of the AVZ.STORAGE definition.

Table 9-2 AVZ.STORAGE definition

| Column            | Description                                     |
|-------------------|-------------------------------------------------|
| PRODUCT_SUBSYSTEM | The four-character DMV subsystem ID.            |
| INTERVAL_START    | The start time of the summary activity.         |
| MAXIMUM_USERS     | The maximum number of users allowed.            |
| SUBPOOL           | The name of the virtual storage information.    |
| BELOW_16M         | The amount of memory in use under 16 Megabytes. |
| ABOVE-16M         | The amount of memory in use over 16 Megabytes   |
| SMFID             | The SMFID as defined within the DVM server.     |

### 9.2.3 Monitoring by using DVM ISPF panels

The DVM server tracks a subset of system resources and can be viewed by using option F4 on its ISPF interface.

The DVM ISPF interface (see Figure 9-3) provides more information about the type of request that is coming into the DVM server by tracking the source of these inputs, the time it takes to run, and so on. This information is useful and helps administrators to understand workloads that are running on the DVM server.

```

----- Server Activity -----
Option ==> █
 1 Interval Activity - Server CPU Interval Summary
 2 Remote Users - Monitor Remote Users
 3 Remote Users by Group - Monitor Remote Users by Group
 4 Storage Monitor - Virtual Storage Utilization
 5 Task Monitor - Active Product Tasks
 6 Statistics - Current Product Statistics

```

Figure 9-3 DVM ISPF panel

## 9.3 Capacity planning for future growth

Although they are not exact numbers, mainframe capacity planners find CPU hours, MIPS, and MSUs are good indicators for the ability to hand workloads.

Organizations should consider hardware upgrades, use of zIIP and zAAP specialty processors for processing improvements and optimal performance.

Much of the planning for the DVM server has more to do with configuration and use, rather than entirely on increases in existing or additions of new workloads. To determine the optimal number of servers that is needed to manage workloads, an iterative approach is recommended, in which the number of DVM server instances can require adjusting. The number of DVM servers generally ranges 1 - 3 for an environment, based on the workload demands.

### 9.3.1 Customer example

A customer needed to validate if their DVM server can sustain a transactional workload by using a JDBC connection to meet subsecond response time objectives. The scenario resulted in configuring each DVM server to run 300 concurrent threads and by managing the SQL rate per thread up to 4,000 statements per second.

This example configuration allowed for the ability to easily scale across multiple servers in their simulated environment.

This example also is a simulated workload that is driven by using an SQL load testing tool, based on the CPU configuration, data (VSAM), and SQL complexity. This example is for reference only and specific to this customer environment and set of requirements.

### 9.3.2 General recommendations

Table 9-3 provides general guidelines to maximize performance for an environment that is running more than one DVM server.

Table 9-3 General recommendations for performance for capacity planning

| Triggers                   | Recommendations                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Number of DVM servers      | <ul style="list-style-type: none"> <li>▶ Perform load balancing with WLM over multiple DVM servers that use the same port</li> <li>▶ Distribute workloads across multiple DVM servers</li> <li>▶ Investigate implementing High Availability that uses VIPA</li> </ul>                                                                                        |
| 8 - 16 Gigabytes of memory | <ul style="list-style-type: none"> <li>▶ Allows:               <ul style="list-style-type: none"> <li>– Better concurrency</li> <li>– Complex SQL to be better processed</li> <li>– Large data pulls to be processed</li> </ul> </li> <li>▶ More memory improves query response time</li> </ul>                                                              |
| Parallelism                | <ul style="list-style-type: none"> <li>▶ Implement Driver or Server-level parallelism</li> <li>▶ Configure MRC and MRCC to run a single request over multiple threads</li> <li>▶ Configure the use of Virtual Parallel Data (VPD)</li> </ul>                                                                                                                 |
| Access method              | <ul style="list-style-type: none"> <li>▶ For keyed data that use Database Control (open thread access is preferred)</li> <li>▶ Open Database Access (ODBA) for concurrent access to data</li> <li>▶ Use IMS direct or DB2 direct for read-only access to bypass related subsystems</li> <li>▶ Distributed Relational Database Architecture (DRDA)</li> </ul> |

| Triggers               | Recommendations                                                                                                                                                                                                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General processors     | Recommend a minimum of two general processors for an initial configuration. Free capacity in GPA buffer to protect against unexpected workload spikes, server outages, or performance regressions in your code. It is recommended to have more free capacity of 30%.                        |
| zIIP specialty engines | Improves latency and transition of GP processing to zIIP processors                                                                                                                                                                                                                         |
| Workloads              | <ul style="list-style-type: none"> <li>▶ Use workload based server definitions (transactional versus large data pulls)</li> <li>▶ Group data sources dictated by the business model</li> <li>▶ Write SQL statements whose result set size does not exceed the LPAR configuration</li> </ul> |

## 9.4 Scaling for growth with the DVM server

The DVM server is confirmed to support over 1,000 tables through internal testing without any effect on local processing or memory. It also successfully tested with up to 15,000 virtual tables. The architecture is designed to perform well for environments with many virtual tables.

DVM for z/OS supports IBM Parallel Sysplex, which is a cluster of IBM mainframes that act together as a single system image. Sysplex often is used for Disaster Recovery and combines data sharing and parallel computing to allow a cluster of up to 32 systems to share a workload for high performance and high availability.

Specific usage for IMSplex is not supported as of this writing. IMSplex is a set of IMS systems working together to share resources or message queues (or both) and workload. Basic support exists for ODBA, which is needed to support IMSplex, but is insufficient for a connection to IMSplex with access to multiple IMS regions.

Database administrators can stop and start servers as needed for maintenance or addressing the need for scalability of workloads. Flexibility also exists for managing across sysplexes or for use in providing a highly available solution.

No significant use or accumulation of resources is associated with DVM servers that require that they be restarted routinely or as part of scheduled maintenance.

The DVM for z/OS architecture can start and run multiple DVM servers concurrently to help address high levels of user concurrency or thread concurrency. The use of multiple DVM servers is easily managed through scripting and can be performed in minutes to help satisfy peak operations or general scalability for business applications.

### 9.4.1 Memory consumption

It is recommended to start 8 - 16 GB of memory and then increase up to 32 GB of memory virtual storage per system for an initial deployment. Allocating memory helps to avoid staging data on each participating IBM Z system. Based on a customer's complexity of SQL, JOINS, and other types of operations that can be performed in memory, more capacity might be needed, depending on current usage.

The number of virtual tables does not degrade performance, but the number of virtual tables that are joined into a complex SQL can exceed the total memory that is allocated to the SQL engine. The amount and length of columns, data types, and the number of records that are materialized in physical memory must be monitored.

The number of SQL statements that are running concurrently depends upon the memory allocated versus the size of the result set record, number of records read, number of users, and so on. Writing optimized SQL statements can reduce the workload that is required for an IBM Z system.

## 9.4.2 zIIP processing

The DVM server can run in Service Request Block (SRB) mode, which allows significant portions of workloads to be offloaded onto zIIP processors (in some cases up to 99%). This feature delivers a low-cost benefit for organizations because they can offload workloads onto much more cost-effective specialty engines in comparison to General Processors.

zIIP eligibility varies based on the access path to underlying data structures, and DVM-specific operations that are performed on that data. Offload also varies across DBCTL, Java, and combinations of z/OS Connect, IMS Connect, and IMS-Direct, for example. Each data source has a specific range of zIIP eligibility.

Consider the following points:

- ▶ IMS DBCTL does not provide any zIIP eligibility.
- ▶ Java offers unrestrained zIIP eligibility by using DRDA.
- ▶ Java workloads that originate outside of the IBM Z system do not benefit from zIIP eligibility.
- ▶ z/OS Connect to IMS Connect delivers a portion of zIIP eligibility.
- ▶ IMS Direct for larger data pulls to support analytics is fully zIIP eligible (non-TXN environment).

IMS operations are general processors that are bound for DBCTL; however, subsequent processing of that data that is run within the DVM server (for example, JOINS) has a degree of zIIP-eligibility. Based on experience, approximately 40% of total SQL operations can be offloaded for zIIP processing.

Because the calling method (DRDA) is restricted to general processes, the zIIP offload is attributed to DVM instrumented code. This processing is separate and independent of any initial IMS operations.

zIIP-eligible workloads use the IMS Direct access method to perform large data pulls. The DVM server processes up to 99% of eligible processing when zIIP processors are available on the IBM Z system. Highly transactional workloads that are running against IMS data perform better by using the DBCTL access method.

When new applications are not at their full productive state, it is difficult to gauge the overall effect. For some time, the workload on IMS increases as existing and new applications run simultaneously. When new workloads are driving highly concurrent threads or users, you might want to add a zIIP processor to avoid any CPU processor contention.

## 9.4.3 Use of MapReduce for parallelism

DVM for z/OS supports parallel processing and can be configured according to the processing needs of an organization. The DVM server supports various types of parallel processing, such as MRC, MRCC, and VPD, at the driver level and server level.

MapReduce does not have the same effectiveness for SQL statements that use Order By or Group By clauses.



A best practice is to establish a controlled benchmark. IBM conducted an internal benchmark for the DVM server and its use of zIIP special engines. Table 9-4 and Table 9-5 list zIIP eligibility and the effect of parallel execution by using MapReduce.

Table 9-4 Sample test results for zIIP usage for virtualized data

| Row label | Sum of CPU time (ms) | Sum of zIIP time (ms) | Sum of IICP time (ms) | Sum of zIIP NTime (ms) | %zIIP eligibility |
|-----------|----------------------|-----------------------|-----------------------|------------------------|-------------------|
| DVM1      | 7099.03              | 5609.55               | 1389.58               | 5609.55                | 98.59%            |

Table 9-5 shows how parallelism allows for the maximum use of GPA and zIIP, which provides organizations with the best use of resources for persisted data with no changes to application or environment.

- ▶ Test cases 4 and 2 in Table 9-5 feature similar configurations.
- ▶ The degree of parallelism of 8 reduces the overall elapsed time from 89.68 minutes to 17 minutes.
- ▶ Adding three zIIP specialty engines that are shown in Test Cast 8 reduces the elapsed time to 13.82 minutes. This change results in an 85% improvement in query execution time.

Table 9-5 Effect of parallelism on the elapsed time

| Test case | GPP | Amount of zIIP engines | Degree of parallelism | Elapsed time (in minutes) | SMT |
|-----------|-----|------------------------|-----------------------|---------------------------|-----|
| 1         | 8   | 8                      | 8                     | 118.96                    | 1   |
| 2         | 8   | 5                      | 0                     | 98.68                     | 1   |
| 3         | 8   | 5                      | 4                     | 27.05                     | 1   |
| 4         | 8   | 5                      | 8                     | 17.14                     | 1   |
| 5         | 8   | 5                      | 8                     | 20.84                     | 2   |
| 6         | 8   | 5                      | 10                    | 17.00                     | 2   |
| 7         | 8   | 5                      | 16                    | 15.73                     | 2   |
| 8         | 8   | 8                      | 8                     | 13.83                     | 1   |
| 9         | 8   | 8                      | 8                     | 17.62                     | 2   |
| 10        | 8   | 8                      | 16                    | 11.72                     | 2   |

#### 9.4.4 Performance differences by access method

DVM for z/OS supports various types of access methods when connecting to target data sources. Organizations can choose the suitable access method based on transactional needs.

If a full load of Db2z is required, assess the ability to perform a Db2 direct read, which results benefits in terms of speed (elapsed time) and zIIP utilization because it is SRB enabled. Table 9-6 on page 206 lists some sample timings that are based on IUD operations that use the Integrated Data Facility (IDF) access method with DVM. The results in the table show some variation regarding access methods to mainframe and non-mainframe data sources.

Table 9-6 Sample access methods, performance impacts with the same degree of parallelism, GPA, and zIIP resources

| Access           | Insert      | Update     | Delete      |
|------------------|-------------|------------|-------------|
| IMS (DBCTL)      | 99.93443898 | 99.3218316 | 99.21321732 |
| IMS (OTT)        | 99.95473499 | 99.3764378 | 99.36438997 |
| DB2 pass-through | 99.32632198 | 99.3476346 | 99.38136378 |
| VSAM             | 84.62641988 | 77.337337  | 81.2397289  |
| Jgate - Oracle   | 97.83463378 | 97.2623789 | 97.32113628 |

## 9.5 Workload balancing

Inbound connections are automatically directed to the DVM server instance that has the most available resources. To determine which instance handles a request, the DVM server evaluates the number of connections that are handled by each instance, and the availability of virtual storage (over and under the 16-MB line).

Load balancing is not apparent to the client application. An application uses a port number to connect to an instance. Then, the instance determines whether it or a separate instance is better equipped to handle the session. If another instance is a better choice, the session is transferred.

## 9.6 User concurrency

The solution can start and run multiple DVM servers concurrently to help address high levels of user concurrency or thread concurrency. This operation is easily managed through scripting and can be performed in minutes to help satisfy peak operations or general scalability for business applications.

This client achieved their objective with each DVM server running up to 300 concurrent threads before measurable degradation began to occur. This customer also varied the SQL rate per thread up to 4,000 statements per second, which met the customer response time requirements. By using this approach, they easily scaled across multiple servers.

This simulated workload was driven by using an SQL load testing tool. It also was based on the CPU configuration, data (VSAM) and SQL complexity. Other factors can be significantly different from what other customers experience.

These numbers are for reference only and might not be similar to results that other customers achieve.

## 9.7 Best practices for deploying the DVM server

After a successful proof of concept, the next phase involves identifying a test and development environment in which you can realize the power of DVM.

### 9.7.1 Data sources

From a data virtualization architecture perspective, the use of DVM requires knowledge of the data sources, data format, and access methods to virtualize data on disk. The following building blocks are used for progressing your idea to productive use in your organization:

- ▶ Access to:
  - Data sources
  - Copybooks
- ▶ Naming conventions
- ▶ Test data

List all of the data sources that you need to fulfill the requirement. For more information about conducting a Project Survey, see Appendix A, “Project survey” on page 225.

This survey approach is a good way to ensure that you are considering all aspects of data virtualization. After you produce a list of data sources, ensure that you have read/write access to the data.

For data sources that do not have their own metadata, such as flat files or VSAM files, you need a copybook to map the data. These copybooks are always stored in separate libraries and you need access to these libraries. Read access is enough to get started.

### 9.7.2 Naming conventions

Establishing naming conventions for the objects that you are going to create also is important. Consider that objects are likely to exist in several different environments with like-named data sources and schema. You must distinguish between to which environment each object belongs.

Also, naming conventions are important when migrating objects into a production environment. The application developers can provide standards and conventions for their particular business applications.

You also need enough test data for development purposes that balances out production-like samples for the amount and variety of data on which applications perform SQL operations. Ensure that the data that is used is of high accuracy. Many times, it is possible to obtain a subset of masked production data to initially populate your environment. This process ensures that your test environment emulates some key characteristics of your system of record (SOR).

## 9.8 Developing queries

After you set up your environment, you can start developing your queries. The DVM server is ANSI SQL-92 compliant. Future releases will iterate toward compliance with ANSI SQL-99.

For example, the following query successfully runs by using the DVM server:

```
SELECT * FROM table LIMIT 10;
```

However, a similar query using the TOP operator is not currently supported by the DVM server.

```
SELECT TOP 10 * FROM table;
```

### 9.8.1 Creating virtual tables

Virtual tables are the data objects that are needed to map to the data. If your data source is Db2 for z/OS, the DVM server uses the Db2 catalog to map the data. If your data source is VSAM or a flat file, a copybook is needed to map the data.

Many times, system programmers know the copybooks, their layout, and where they map to data on disk. For more information about how to create virtual tables by using DVM Studio or the ISPF interface for DVM, see Chapter 7, “Managing and monitoring” on page 131.

### 9.8.2 Combining data from different data sources

The DVM server can combine different data sources into a single data source. Each of the data sources is defined by a virtual table that maps to the location and data. By creating a virtual view, you can combine individual virtual tables into a uniform and fully transformed set of records. This view can be defined by the selected columns and joined by a unique ID that is common to both data sets. Then, a query predicate can be applied to the definition of the view to filter or reduce the result sets according to the operators in your structured query language (SQL).

You can select a base Virtual table to include as part of a new virtual view, and then modify the DDL for the new view to include all other virtual tables. You also can check to validate the DDL syntax before finalizing your new Virtual View.

### 9.8.3 Creating a query

After you create the objects that you need (a virtual table, virtual view, or Db2 table), you can easily create a query from them by using DVM Studio. The query can be adapted to be less or more restrictive.

You can determine which columns and the subset of data are needed for extraction. Nearly all extract, transform, and load (ETL) data workflow can be written by using SQL syntax for the selection, cleansing, and curation of data. In many ways, the use of predefined data models in the form of virtual tables and views allows for greater sophistication in your design.

## 9.8.4 Testing

Queries can easily be tested to verify your results in advance in DVM Studio or as part of an application. You can test the query directly in DVM Studio by using one of the following methods:

- ▶ Select the query on the panel and press **<F5>**
- ▶ Select the query and then, right-click and select the **Execute SQL** option.

## 9.8.5 Embedding your query in an application

After your query is ready for a functional test, it can be embedded into any application, such as Java. DVM can generate Java code snippets that are based on your query, which can then be pasted into a Java application and then, compiled and run.

Right-clicking the virtual table or virtual view shows the Generate Code from SQL option (see Figure 9-4).

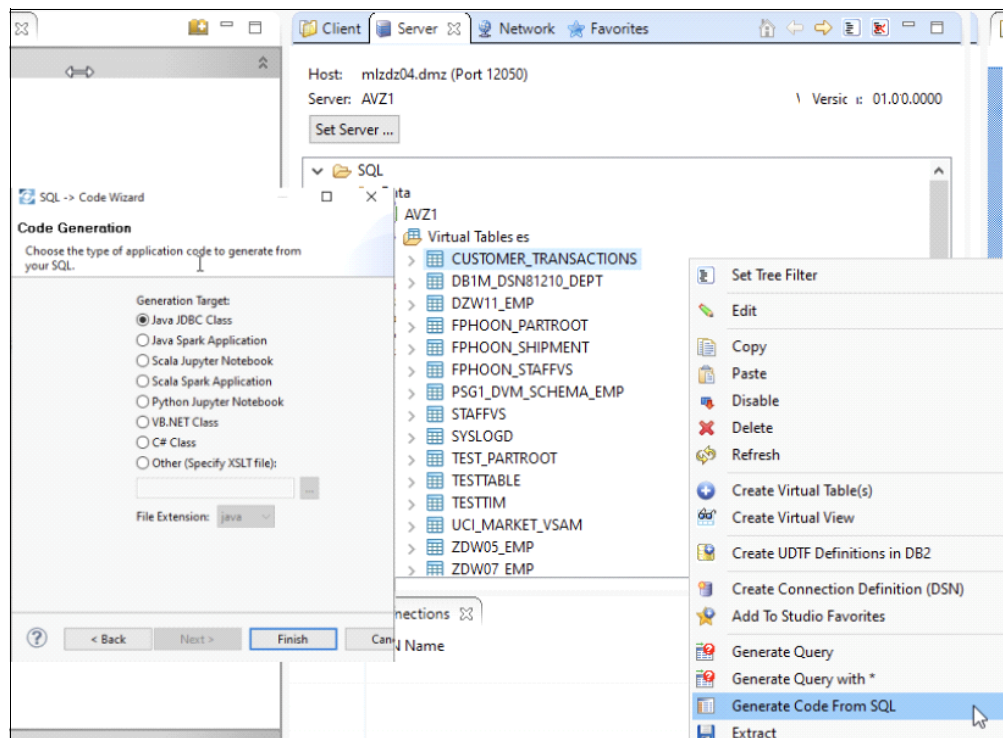


Figure 9-4 Generating code from SQL option

DVM Studio adds to programmer productivity by allowing users to automatically generate and modify queries. DVM Studio provides the ability to compose code snippets for generated or modified queries into a range of programmable APIs, which speeds up the time to prototype applications for development and testing purposes.

Code can be generated in any of the modern programming languages that are shown in Figure 9-5.

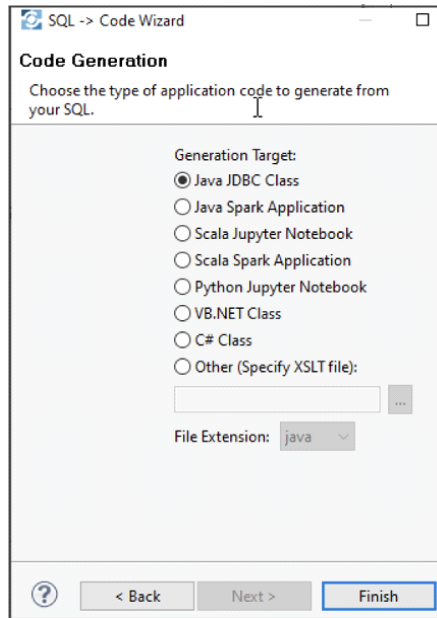


Figure 9-5 Code Generation window in DVM Studio

The generated code includes associated object handling, Java classes, the defined SQL statement, and JDBC connection string with user credentials for successful execution. Generated code can be tested immediately in DVM Studio by placing the cursor anywhere in the Java code and then, right-clicking and selecting **Run as** → **Java application**. The result is shown in the Console view (see Figure 9-6).

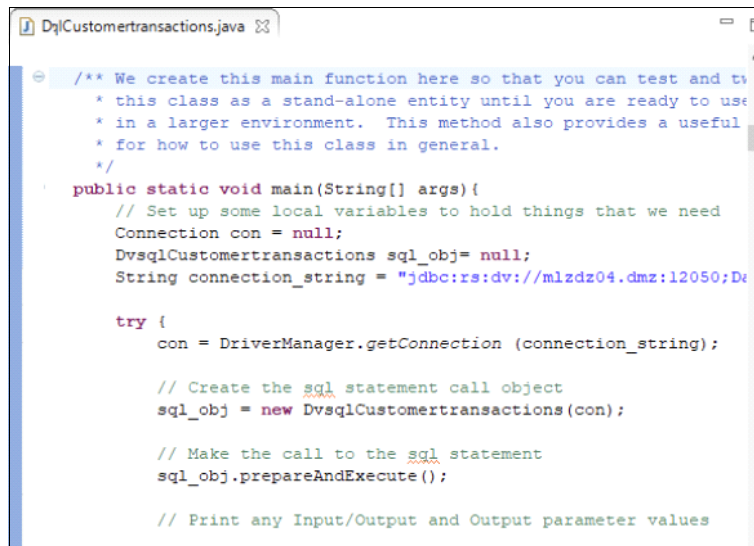


Figure 9-6 DVM Studio generated Java code

The quality assurance phase of deployment is to ensure that the solution you want to implement is not going to harm your production environment. It is a necessary step before finished products are promoted into production.

Deploying applications in the test environment requires that users have suitable credentials and access to data sources. Also, it is important that the required metadata from the DVM server that is associated with virtual tables and views represent the original data from the development environment.

**Note:** If a different user uses this code than when the Java code was created, the Java code must be regenerated by using the correct user credentials.

It is recommended that your test environment mimics your production environment as much as possible. Therefore, the test data must be identical to the production environment with data obfuscation or masking as needed to ensure suitable data privacy and protection of potentially sensitive data.

Moving applications into production is no different than moving an application from development to test. However, both DVM administrative tasks and requirements around high availability often are critical to the reliability and resiliency of business-critical applications.

## 9.9 Administering the DVM server in production

A bigger concern for organizations revolves around providing seamless access to underlying mainframe data. The DVM server controls access by using RACF (or any other security module you use on IBM Z). The DVM server captures information for each user who is attempting access to the mainframe data sets. It also controls access for users who are attempting to access non-mainframe data sources.

### 9.9.1 Limiting access to the DVM server

The client user ID that is accessing the DVM server is like any RACF user ID with data access. The main tasks within the DVM server in a production environment are to create and maintain metadata for production data sources (by a DBA), and perform ongoing monitoring of the IBM Z system (system programmer).

### 9.9.2 High availability configurations

When you are running a parallel sysplex in your production environment on IBM Z (usually with Db2 for z/OS in data sharing mode), you must adapt your installation to accommodate the complexities for the following components:

- ▶ Multiple DVM server instances

Each LPAR within your Parallel Sysplex might need to have an active DVM server because individual servers cannot connect to other servers without extra configuration. Each DVM server must be connected to the local Db2 subsystem; for example, if you use Db2 for z/OS.

- ▶ Shared resources

All resources that are required by the DVM server (data sources, copybooks, configuration files, and metadata catalogs) must be on a shared DASD. In this way, all the DVM servers within your Parallel Sysplex can use the same information.

- ▶ WLM and DVIPA connections

In a production environment, it is recommended to connect your DVM servers to the workload balancer you use (for example, Workload Manager) and enable DVIPA connections to each of your DVM servers. This configuration allows WLM to balance the workload between the servers in your Sysplex. It also allows the workflow to continue uninterrupted if the servers in your Sysplex fail, or a complete LPAR goes offline for any reason.





## Best practices for project success

This chapter provides best practices for defining a successful validation of Data Virtualization Manager for z/OS (DVM for z/OS) software as part of solving critical business problems, which is specific to the use of data virtualization to provide seamless SQL access to data that is on IBM z Systems®.

The chapter works through project definition and best practices and includes the following topics:

- ▶ 10.1, “Defining successful projects” on page 214
- ▶ 10.2, “Defining the approach” on page 214
- ▶ 10.3, “POC checklist” on page 216

## 10.1 Defining successful projects

A successful roadmap helps to balance precarious project elements, such as time and skill against complexity and risk. To motivate technology, you must define an approach that progresses value as you work to bring new technology forward (see Figure 10-1).

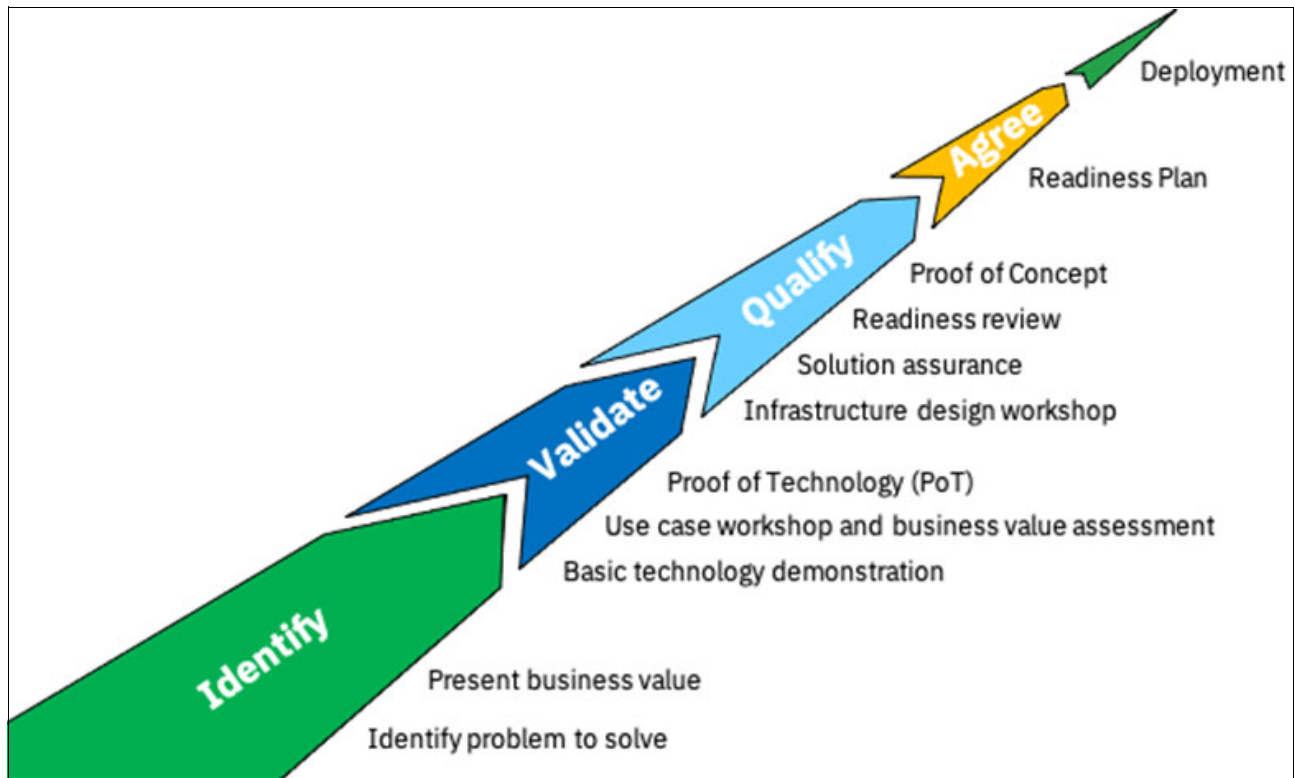


Figure 10-1 Project lifecycle

Setting timelines, scope, focus areas, and success criteria are “must-haves”.

## 10.2 Defining the approach

The best approach for most technology is to funnel the business value through a series of discussions with a balance of business and technical involvement. Technology influences the business decision and ultimately the business decides whether to invest. Starting small and building on interest and business value ultimately leads to a successful implementation.

IBM provides key data assets on its IBM Demos website that help users to explore, learn, and try various IBM solutions. DVM for z/OS includes a guided tour and several videos that are presented by technical experts that walk you through common use cases, highlighting key capabilities.

Figure 10-2 shows the [landing web page](#) for IBM Data Virtualization Manager for z/OS.

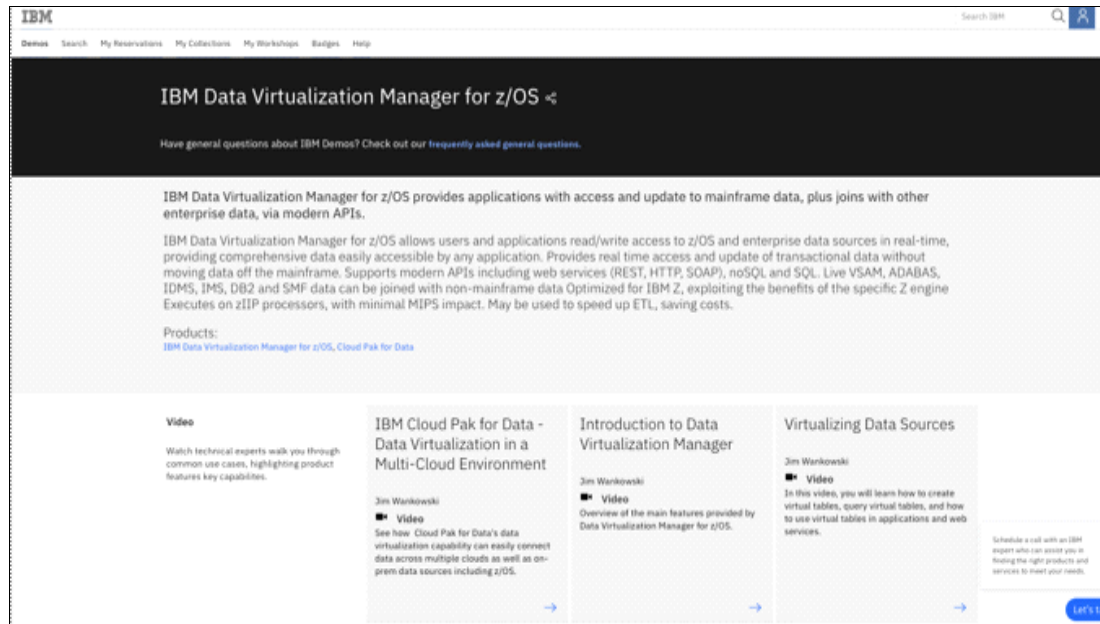


Figure 10-2 IBM Data Virtualization Manager for z/OS landing page

IBM offers a full range of approaches where you can learn more about DVM for z/OS, such as self-service demos, deep dive discussions with product experts, and proofs of technology or concept.

Contact your IBM client representative to schedule a demo or to get more information about DVM for z/OS. Consider the following points:

- ▶ IBM Demos provide a set of useful videos, product tours, and hands-on labs that simplify and offer the ability to exercise technology without levels of investment that are required for local infrastructure, capital, and operating expenses.
- ▶ Deep dive presentations by IBM and Business Partners help to orchestrate a lower-level discussion that describes how DVM for z/OS matches well for their environment. These presentations are a perfect time to uncover the primary use cases.
- ▶ Deep dive Demos by IBM and Business Partners are well equipped to demonstrate various functions.
- ▶ Proof of Technology

IBM has internal lab environments that can be used to demonstrate capabilities and make available on-Z and off-Z sources by using modern APIs, such as SQL, JSON, HTTP, and REST.
- ▶ Proof of Concept (POC)

Not every project requires a fully invested project plan to prove the value to the business. A real concept requires that the business map a project with key milestones, systems, data, infrastructure, and human capital to become realized.

## 10.3 POC checklist

Any project is an investment commitment and an agreement between groups with a mutual goal in mind. Focus on a simple checklist to drive the concept forward that includes the following components:

- ▶ Clearly defined use case
- ▶ Identified sponsors
- ▶ Finances and budget
- ▶ Clear responsibilities
- ▶ Project management
- ▶ Clearly fenced milestones
- ▶ Prevent scope creep
- ▶ Compelling outcome

### 10.3.1 Timelines

When defining timelines for the project, consider the following best practices:

- ▶ Target 90 days maximum for proving your project.
- ▶ Use weekly checkpoints, which are critical to ensure that no issues exist.
- ▶ Keep the pulse of key stakeholders throughout the project schedule.
- ▶ Have successful milestones.

### 10.3.2 Setting scope

When setting scope, ensure that you adhere to the following best practices:

- ▶ Keep the project well defined.
- ▶ After engaging with IBM to set up and run a proof of technology or proof of concept, IBM ensures a successful engagement by capturing goals, objectives, data sources, and uses through a document of understanding, whose framework is shown in Figure 10-3 on page 218.

Remember, Scope creep is the number 1 reason why projects fail.

### 10.3.3 Focus

When defining the focus, keep it on track. Consider the following points:

- ▶ Focus on the business use case.
- ▶ Technology is important specific to the configuration.
- ▶ View the project entirely as a solution, not as a “product”.

### 10.3.4 Success criteria

Well-defined success criteria is critical in knowing whether you accomplished what you set out to do. Consider the following points:

- ▶ Keep it Super Simple (KISS) is a perfect model to follow.
- ▶ Keeping the project simple in scope concisely establishes reasonable goals and timely closure.
- ▶ Success should not be *only* performance-based.
- ▶ Provide an executive summary.

### 10.3.5 Best practices

Consider the following best practices as you go through the proof of concept activities:

- ▶ Incorporate measures from an alternative or today's state of business for comparison with DVM.
- ▶ Over-communicate the project scope and estimate the required effort.
- ▶ Avoid poorly documented DoUs, which lengthen and introduce ambiguity into the project.
- ▶ Maintain regular checkpoints to keep the project on track and heading in the right direction.
- ▶ Monitor performance through SMF to validate “do no harm” effects on production.
- ▶ Ensure that your data sources are accurate and current by avoiding staging or copying data.
- ▶ You are on track if you start with simple access and query results.
- ▶ Be flexible. If the initial use case fails, it is typically followed by other uses that have positive effects.
- ▶ Success can breed scope creep; therefore, stay on track and close out the current project.
- ▶ Be sure to really rationalize the use cases. A poorly defined use case can deflate the project.

As shown in Figure 10-3, an agreed framework helps to drive an overall comprehensive approach for a proof of concept. The agreement features specific sections that capture all elements from requirements, use cases, business drivers, and closing criteria.

| DoU Framework                                                    |                                                                                                        |
|------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <a href="#">1.0 INTRODUCTION</a>                                 | <a href="#">APPENDIX A – TEST SPECIFICS / CUSTOMER ENVIRONMENT</a>                                     |
| <a href="#">2.0 MISSION STATEMENT</a>                            | <a href="#">A.1 IBM Software Products</a>                                                              |
| <a href="#">3.0 COMPANY BUSINESS OBJECTIVE</a>                   | <a href="#">A.2 Data &amp; Application Connections</a>                                                 |
| <a href="#">4.0 COMPANY BUSINESS REQUIREMENTS</a>                | <a href="#">A.3 Hardware Systems</a>                                                                   |
| <a href="#">5.0 SOLUTION ARCHITECTURE</a>                        | <a href="#">A.4 Network Topology</a>                                                                   |
| <a href="#">6.0 MAJOR TASKS AND RESPONSIBILITIES</a>             | <a href="#">A.5 Preparatory Questions</a>                                                              |
| <a href="#">7.0 PROPOSED TESTS AND CRITICAL SUCCESS CRITERIA</a> | <a href="#">APPENDIX B – CUSTOMIZED POC SCENARIO DETAILS</a>                                           |
| <a href="#">8.0 PROJECT ASSUMPTIONS</a>                          | <a href="#">B.1.1 Data Sources – Online access USE CASE USING Data Virtualization Manager for z/OS</a> |
| <a href="#">8.1 Human Resource Assumptions</a>                   | <a href="#">B.2.1 Functional Tests</a>                                                                 |
| <a href="#">8.2 Technical and facilities Assumptions</a>         | <a href="#">B.2.2 Non-Functional Tests</a>                                                             |
| <a href="#">8.3 Customized POC location</a>                      | <a href="#">B.3 Data sources mapping components (Copybooks, Target Environment)</a>                    |
| <a href="#">8.4 Preparation Meeting</a>                          | <a href="#">APPENDIX C – INSTALLATION CHECKLIST</a>                                                    |
| <a href="#">8.5 Roles and Responsibilities</a>                   | <a href="#">C.1 Logical Diagram and S/W Components</a>                                                 |
| <a href="#">9.0 TECHNICAL MILESTONES</a>                         |                                                                                                        |

Figure 10-3 DOU framework example

### 10.3.6 Roles and responsibilities

Failure to define roles and responsibilities (as listed in Table 10-1) is the downfall of any project. Without the “who” and the “what”, success is hard-fought. With this idea in mind, assign a name to each role on the project team. Identify at least one project manager and ideally two that complement or bridge the technical team to the respective line of business. With these roles assigned, the scope, project velocity, key milestones, and results are always visible.

Table 10-1 Roles and responsibilities

| Role                              | Responsibilities                                                                                                                                                                                 | Person                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| Project manager (business)        | Provide business physical and technical resources to complete the objectives of the POC. Facilitates scheduling of technical resources, timeframes, POC technical goals, and objection handling. | David                 |
| Use cases (business)              | Participate and support use cases execution and evaluation.                                                                                                                                      | Suali                 |
| System administration (business)  | Responsible for DVM installation, configuration, and monitoring (including authentication, authorization, and network tasks).                                                                    | Fred, Susan           |
| Database administrator (business) | IBM Db2 for z/OS experts or those team members who are responsible for DVM integration with Db2 for z/OS.                                                                                        | Ankar, Lei Ping       |
| Application developer (business)  | Participate and support Copybook – VSAM mapping                                                                                                                                                  | Ana, Denis, Francesco |
| Coordinator (technical team)      | Primary customer contact                                                                                                                                                                         | Marie                 |

| Role                                | Responsibilities                                                                                                                                                                                            | Person       |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| Project manager (technical team)    | Provide technical resources to complete the objectives of POC. Facilitates scheduling of technical resources, timeframes, and objection handling.                                                           | Jonathan     |
| Product specialist (technical team) | Technical contacts providing on-site or remote technical support for business POC. Help to install and configure DVM and to assist use case execution and evaluation. Provide technical knowledge transfer. | Andrew, Bill |

Define a detailed timeline and project plan to maintain periodic status calls and checkpoints. The success or failure of your project might depend on this frequency. A weekly call for 30 minutes at a time is recommended to help speed up the resolution of issues.

### 10.3.7 Installation

Installation of the DVM SQL engine and administration Studio is simple and straightforward. IBM DVM for z/OS SMP/E installation is quick and DVM Studio is an MS-Windows point-and-click installation that is fast and easy.

### 10.3.8 Configuration

During the configuration process, consider the following tips:

- ▶ Always perform IVP sample VSAM virtualization by using DVM Studio.
- ▶ Upon start, be sure to read the DVM started task SYSOUT carefully.
- ▶ Only configure what you need:
  - Change or activate only the required IN00 sections.
  - JGATE does not need to be configured if no external or relational data sources are involved with the project.
  - Db2 for z/OS does not need to be configured if it is not part of the project, regardless of the benefits that are realized by IDF or Db2 UDTF.

### 10.3.9 Architectural topology

The architectural topology (see Figure 10-4 on page 220) must be discussed with your project teams and up-line business owners to ensure that you have a schematic that represents reality. Working with Systems and IT operations team members, build out a facsimile of the current environment.

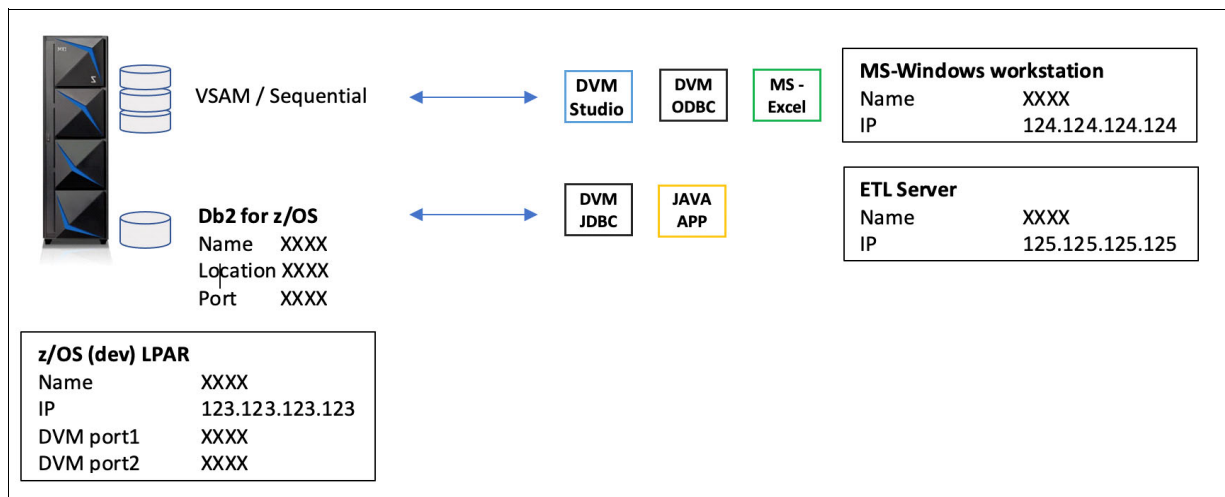


Figure 10-4 Sample topology

Also, the project team should ensure that all endpoint client access is clearly identified with the associated access method and data flow, including all staged and non-staged read/write activity.

### 10.3.10 Defining use cases

When defining use cases for your project, put more emphasis on functional aspects than performance-driven characteristics. Instances exist in which performance metrics are difficult to measure.

You also want your project to be configured in a non-production environment if at all possible. If the number of zIIP processors is limited, DVM workloads can be run by General Processor engines. If performance testing is unavailable, work to ensure that your tests are measurable and well-defined.

Limit the number of use cases to two if possible and allow for up to two separate tests for each use case; for example, use two Db2 for z/OS tables and two VSAM virtual tables. Remember that your project is not a surrogate for production-ready development and testing.

When working with VSAM and sequential file sets, limit the scope to the number of virtual tables, instead of data segments, because hundreds of copybooks can exist that are associated with a single VSAM file. Application developers must be involved in your project and sample copybooks and data are the first requirement to validate the data set and determine any invalid data or data types.

Possible use cases include the following examples:

- ▶ **Client Access:** Typically, businesses focus the POC on how the DVM server can satisfy needs for critical business applications. They see the implementation is the primary target. After the technology is acquired, customers gain experience by solving lower tier business problems and progress toward meeting the more substantial ROI for the larger and more critical applications that inspired the POC.

You can shorten the testing intervals for a project by using common tools and utilities. In these cases, use the ODBC/JDBC SQL drivers. Also, consider the use of the DVM server as a complement to ETL processing by greatly improving parallelism and reducing the overall latency of copying subsets of transformed data to an operation store for downstream processing or reporting.



- ▶ Data integration: Use a maximum of two virtual tables for extracting large data sets that use the JDBC driver. Consider the use of DVM for z/OS as a data pump for ETL tools.
- ▶ Read/write access: DVM for z/OS also can WRITE back to original data sources. This method is a good way to simulate transaction-based activity by using DVM Studio.
- ▶ IBM Cloud Pak for Data: Use the DVM connection service with DV, Watson Knowledge Catalog, or Watson Studio to experience the full power of IBM's new platform architecture that is built for the Hybrid Cloud.
- ▶ DVM as an application server: Use DVM as an application server by way of IDF for DRDA access to Z data

### 10.3.11 Best practices for defining success criteria

When defining the success criteria for your project, consider the following best practices:

- ▶ Usability: Ease of use for DVM installation and configuration (up and running user experience).
- ▶ Data access by way of Studio: The ability to access, transform, and virtualize VSAM and sequential files, and Db2 for z/OS tables by using DVM Studio.
- ▶ Data access by way of xDBC drivers: Ability to access, transform, and virtualize VSAM and sequential files, and Db2 for z/OS tables.
- ▶ Handling loads or data synchronization for large amounts of data. Establish an evaluation of performance (sequential versus parallel) for bulk copies of data.

### 10.3.12 Concluding the POC

After your project completes the evaluation phase, the project should result in a clear set of tangible assets that speak to the original problem statement, milestones met metrics, and results that clearly articulate the value proposition and effect for the business.

### 10.3.13 Finalizing deliverables

A first step in finalizing the deliverables is to verify acceptance with your stakeholders or project sponsors and determine whether any extensions or more testing during deployment must be amended to the current agreement for the project (the DOU framework).

Create a full disclosure report and present it to key decision-makers that includes the following information:

- ▶ Summary of the project scope
- ▶ Clearly reemphasizes the success criteria and results obtained
- ▶ Visuals (performance charts, metrics, and so on)
- ▶ Customer user feedback (including direct quotes)

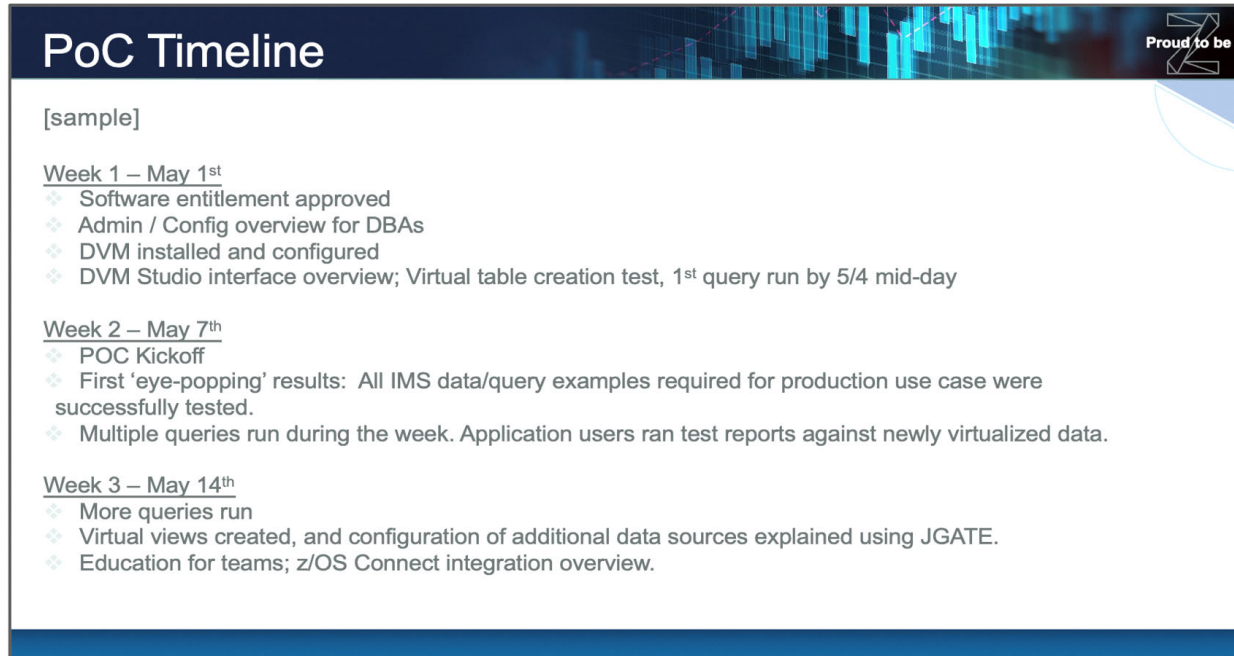
Review the report with the technical sponsors for the project before delivering it to key stakeholders in a larger audience. Work to ensure that all agreement details were met for the overall sentiment of success criteria and results.


#### Elements of the full disclosure report examples

On this section, we present examples of elements of the full disclosure report that give you an idea of how the report might be built.

### Project timeline

Figure 10-5 shows a sample project timeline, including weekly actions and results.



**PoC Timeline** Proud to be 

[sample]

Week 1 – May 1<sup>st</sup>

- ❖ Software entitlement approved
- ❖ Admin / Config overview for DBAs
- ❖ DVM installed and configured
- ❖ DVM Studio interface overview; Virtual table creation test, 1<sup>st</sup> query run by 5/4 mid-day

Week 2 – May 7<sup>th</sup>

- ❖ POC Kickoff
- ❖ First 'eye-popping' results: All IMS data/query examples required for production use case were successfully tested.
- ❖ Multiple queries run during the week. Application users ran test reports against newly virtualized data.

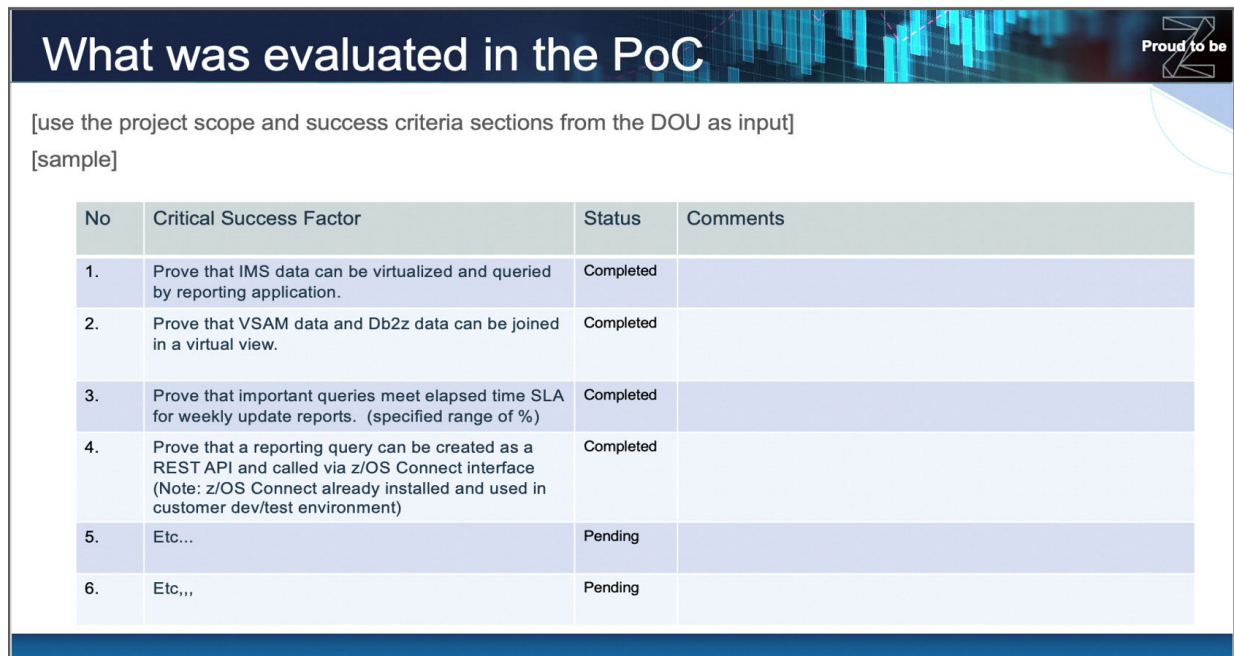
Week 3 – May 14<sup>th</sup>


- ❖ More queries run
- ❖ Virtual views created, and configuration of additional data sources explained using JGATE.
- ❖ Education for teams; z/OS Connect integration overview.

Figure 10-5 Project timeline

### Evaluation of critical success factors and status

Figure 10-6 shows an example of the report that highlights the success criteria and results obtained.



**What was evaluated in the PoC** Proud to be 

[use the project scope and success criteria sections from the DOU as input]

[sample]

| No | Critical Success Factor                                                                                                                                                          | Status    | Comments |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|----------|
| 1. | Prove that IMS data can be virtualized and queried by reporting application.                                                                                                     | Completed |          |
| 2. | Prove that VSAM data and Db2z data can be joined in a virtual view.                                                                                                              | Completed |          |
| 3. | Prove that important queries meet elapsed time SLA for weekly update reports. (specified range of %)                                                                             | Completed |          |
| 4. | Prove that a reporting query can be created as a REST API and called via z/OS Connect interface (Note: z/OS Connect already installed and used in customer dev/test environment) | Completed |          |
| 5. | Etc...                                                                                                                                                                           | Pending   |          |
| 6. | Etc,,,                                                                                                                                                                           | Pending   |          |

Figure 10-6 Evaluation report

### ***Business value***

The business value of the example project is highlighted by the following goals that were accomplished:

- ▶ Access to combined relational and non-relational data sources on z/OS in real time for downstream analytics and reporting. React to business demands quickly:
  - Offload VSAM data and merge with Db2 LUW data that used to take two days end-to-end.
  - The daily summary report now occurs in real time with less than 5 minutes of transaction latency.
- ▶ The use of existing infrastructure, processes, and people:
  - Eliminated VSAM data movement to x86 environment and save time and personal hours
  - No extra infrastructure cost was required to access data in place, while maintaining SQL compatibility
- ▶ Provide real-time access for BI solutions instead of working with of outdated, inaccurate non-Z data:
  - BI solutions and IBM Z solution can all see real-time data
  - Access is fast and cost-effective
  - Greater than 95% offload to zIIP processors with lowered costs

### ***Business and sponsor quotes***

Be sure to capture positive statements by business stakeholders as you progress through your project. The bottom line is often driven by the positive perceptions that are captured. TCO and savings always are critical factors in the success of a project.

Also, performance places a critical factor, but always incorporate the user experience and overall usability of your solution. Collect the following types of outtakes from your project that and present them to the business:

- ▶ Setup is simple and straightforward for those with TSO access.
- ▶ Mainframe access was immediate.
- ▶ User experience was identical regardless of data source.
- ▶ Benefits that are derived from zIIP specialty engine offload without having to create special processing.





# A

## Project survey

The information and tables in this appendix help you to capture information that is related to your z/OS environment and workloads as it relates to processing, storage, and memory impacts on an environment as part of running DVM for z/OS.

This appendix includes the following topics:

- ▶ “Business drivers” on page 226
- ▶ “Virtualization topology” on page 227
- ▶ “Primary use cases” on page 227
- ▶ “Physical storage or memory” on page 228
- ▶ “Environment” on page 229
- ▶ “Access to data sources” on page 230
- ▶ “Application workloads” on page 231

## Business drivers

Data Virtualization Manager for z/OS (DVM for z/OS) delivers a virtualization architecture for accessing z/OS and non-z/OS data sources. It also performs transformative operations that translate hierarchical data formats into a relational format for SQL-based and modern applications to access. The program generates virtual tables and virtual views that logically map to files, segments, and data records that are on an IBM Z LPAR environment or distributed database servers.

The data virtualization service brings great value for generating shared access to “difficult to access and work with” data that is on IBM Z. DVM for z/OS dramatically reduces the work that is associated with ETL operations by allowing direct real-time access to persisted data. This data type represents the Source of Record for many critical business applications.

DVM for z/OS supports z/OS and non-z/OS data sources and can deliver scalability, reliability, and extreme performance through its unique machine-specific optimization. DVM also supports read and write SQL operations in parallel by using MapReduce.

DVM delivers zIIP-eligibility, which works to offload MIPS from General Processors that in turn reduce the cost for running production applications. DVM lowers total cost of ownership (TCO), provides high return on investment (ROI), and fast time to market (TTM), which are critical for driving business agility.

Service level agreements or objectives also influence infrastructure, topology, and overall design decisions for data virtualization needs. In many instances, downtime is an inhibitor to introducing new technology to a technology stack.

One of the strengths of this solution is its simple and flexible deployment of the solution, which is resident to the IBM Z. The program features high availability capabilities that ensure resiliency and improved fault tolerance when network, software, and hardware failures occur. It also facilitates update or refresh software functions by allowing for zero downtime for software refreshes or fixes.

Ask the following questions:

- ▶ What are the primary pain points for the business when it comes to accessing z/OS data?
- ▶ Is an outage acceptable for the initial setup?
- ▶ What are business requirements for application response times?
- ▶ What are business requirements around resiliency or availability of the solution if an LPAR ceases to function?
- ▶ What are business requirements around User, Group, and Role level security?

## Virtualization topology

The designed solution offers some strengths that allow for added simplicity and flexibility across your IBM-driven solutions for hybrid cloud infrastructure. The solution delivers an IBM Z resident installation that is straightforward with setup and installation in the order of hours, not days.

DVM is IBM Z-resident software that is required for installation to a designated LPAR. Specific configurations that need high availability (HA) requires multiple installations and corresponding licenses to support failover and failback when an outage or disaster occurs.

The solution includes the flexibility to run multiple DVM servers concurrently to aid in overall performance. Therefore, file system requirements must be sized to support a range of DVM servers, based on transactions per hour and the amount of user concurrency.

### Distance between the DVM servers and data sources

The distance between DVM servers that access z/OS or non-z/OS datasets should be considered in the planning phase for any initial installation and configuration.

Ask the following questions:

- ▶ What is the distance between IBM Z in a Sysplex that is participating in data virtualization?
- ▶ If non-local data sources are used, what is the distance between the DVM server and the target database server?
- ▶ What is the distance between database servers that are planned to support HA for the data virtualization environment?
- ▶ What is the current cross-site network bandwidth between these servers today and is that shared with multiple applications or dedicated?

Insufficient network bandwidth can lead to an increase in latency, which increases response and execution time.

- ▶ What type of network configuration is in place between existing systems that are targeted for data virtualization? Also, do specific requirements for security exist, such as SSL?

### Primary use cases

Our data virtualization solution offers support for a range of use cases that range from shared access, centralized management, HA use with Disaster Recovery support through continuous delivery following our initial release.

Use cases continue to surface as the data landscape changes in the market and customer environment. The following examples are of primary use cases in use today:

- ▶ Real-time access to disparate data across z/OS and non-z/OS data sources
- ▶ Modernize mainframe applications to enable web-based and mobile applications for access
- ▶ Centralize access with control to drive governance for a trusted view of all enterprise data
- ▶ Low-level data integration that enables copying data by way of SQL-based operations
- ▶ Use data virtualization as a methodology for application development and incubation of production-level prototypes

- ▶ Establish a single view to drive operational and business analytics
- ▶ Forge new business models by using Machine Learning algorithms and Artificial Intelligence over shared data

Ask the following questions:

- ▶ Do you plan to use data virtualization for workload balancing to offload queries to one or more sources?
- ▶ Do you anticipate the use of data virtualization as part of your build, test, and deployment process for new applications through a logical Data Model that is targeted for development, test, or quality assurance?
- ▶ Do you plan to use data virtualization to reduce business interruption or outages that are caused by planned or unplanned maintenance windows? If so, what are your SLA/SLO ranges across multitier business applications?

## Physical storage or memory

To support data virtualization processing, DVM can use more physical storage or physical memory to achieve the most optimal query run time or write operation. As data transformations, pushdown operations, JOINS, and UNIONS and functions are run, more resources can require specific “pre” and “post” processing of a normal operation.

Ask the following questions:

- ▶ What is the range of daily transactional volume for virtualized data across systems that are targeted for replication?
- ▶ Are you open to using more zIIP processing to help in achieving your performance goals?
- ▶ Would you like to have flexibility around the number of active DVM servers processing read/write operations to drive improved parallelism for generalized workloads beyond local processing occurring independently of the DVM server?
- ▶ Would you like this to be configurable within the user interface upon initial configuration or as part of the management of an active data virtualization environment system while in production state? Is a managed outage or failover to secondary acceptable to perform this operation?



# Environment

Foundational to the use of DVM for z/OS is gaining and understanding of the underlying z/OS system, its configuration, memory, processing power, and security. Each element is critical to the configuration and optimization of virtualized data assets that are managed by the DVM server.

## Hardware configuration

Specify the memory requirements as real storage that is assigned to an LPAR in Gigabytes. If the product is installed on more than two LPARs, specify the largest and smallest amount of configurable memory that can be used.

Table A-1 can be used as a template for capturing your development, test, and production environments.

Table A-1 Hardware configuration

| LPAR | Model | Number of GPPs | Number of zIIPs | Memory (GB) | Comments |
|------|-------|----------------|-----------------|-------------|----------|
|      |       |                |                 |             |          |
|      |       |                |                 |             |          |
|      |       |                |                 |             |          |

## z/OS environment

Specify support software programs that are targeted for use with data virtualization. Add configurations by using Table A-2.

Table A-2 z/OS environment configuration

| Configuration                   | Y/N | Details |
|---------------------------------|-----|---------|
| Security                        |     |         |
| ENQ manager                     |     |         |
| VSAM RLS                        |     |         |
| Innovation IAM                  |     |         |
| CDC for Db2                     |     |         |
| CEC for VSAM                    |     |         |
| z/OS Connect Enterprise Edition |     |         |
| CICS                            |     |         |
| IBM Cloud Pak for Data          |     |         |

## Access to data sources

Taking inventory of the active data sets that are targeted for use with DVM for z/OS is important to understand. In addition to the data sources, it is important to specify the various client connections, types of workloads, and version levels.

### Client connections to DVM for z/OS

Specify applications or client tools that access the DVM server for SQL-based read/write operations by using Table A-3.

Table A-3 Client connection information

| Product                       | Product version | OS | OS Version | REST | JDBC | ODBC |
|-------------------------------|-----------------|----|------------|------|------|------|
| IBM Data Stage                |                 |    |            |      |      |      |
| IBM Query Management Facility |                 |    |            |      |      |      |
| Informatica PowerCenter       |                 |    |            |      |      |      |
| Microsoft Excel               |                 |    |            |      |      |      |
| Microsoft Power BI            |                 |    |            |      |      |      |
| Tableau                       |                 |    |            |      |      |      |

### Data sources

Specify the data sources that must be virtualized and available to the mainframe, Java, ETL, analytic, and reporting tools by using Table 4.

Table A-4 Data sources

| Product                     | Product version | Platform | Data source       | Workload | Platform | ODBC |
|-----------------------------|-----------------|----------|-------------------|----------|----------|------|
| Adabas                      |                 |          | Oracle Cloud      |          |          |      |
| Db2 distributed             |                 |          | Netezza           |          |          |      |
| Db2 for z/OS                |                 |          | Postgres          |          |          |      |
| Hadoop (Apache, HDP,CDH,GP) |                 |          | SQL Server        |          |          |      |
| IDMS                        |                 |          | Azure             |          |          |      |
| IMS                         |                 |          | AWS Redshift      |          |          |      |
| MongoDB                     |                 |          | Sequential files  |          |          |      |
| IBM MQ                      |                 |          | SMF               |          |          |      |
| MySQL                       |                 |          | SYSLOG            |          |          |      |
| Oracle                      |                 |          | Teradata          |          |          |      |
| Oracle Exadata              |                 |          | VSAM or CICS/VSAM |          |          |      |

## Application workloads

Workloads often are separated into online workload or batch workload. Online workload can be divided further by line of business. Active/active workload has more strict definitions. The active workload is a business-related definition and is the aggregation of the following items:

- ▶ Software: A user-written application and the middleware runtime environment.
- ▶ Data: A set of related objects that feature maintained transactional consistency, and optionally, referential integrity constraints preserved.
- ▶ Network connectivity: One or more TCP/IP addresses or hostnames and ports (for example, 10.10.10.1:80).

This definition is intended to preserve the transaction consistency of the data. Data virtualization supports all create, retrieve, update, and delete operations that are supported by the underlying data sources. This support includes data type mapping, function mapping, and optimized transformations into virtual tables and virtual views.

The DVM solution receives client requests, performs costing calculations and parsing operations to best optimize the round-trip response and execution time for a workload.

Applications can be select subsets of data or a complete data set through a subset of virtual tables or virtual views within or across schema. Processed workloads can have multiple query plans that include transform, pushdown, and JOIN operations in specific frequency and volume daily. These applications are critical to business operations and this solution works to optimize query execution and response time for requesting applications.

Ask the following questions:

- ▶ What types of application workloads are active in your environment (batch, transactional, ISPF, and so on)?
- ▶ Do you have a maintenance window for deploying application upgrades to the production system during which you take the server offline?
- ▶ Do you perform DDL operations for specific workloads or have write-intensive applications?
- ▶ How frequently do you delete data (and how much)? Truncate? Entire table? Subset with a where clause? How do you typically load data or perform bulk operations (for example, external table)?
- ▶ Do you use indexes or primary keys for your data models?

Complete the table that is shown in Table A-5 for the best representation of critical workload types. A sample entry is provided in the first row of Table A-5.

Table A-5 Workload information

| Workload name         | Volume of data                                | Transaction rate                                                         | Workload characteristics                                                  | Responsive objectives               | Execution objectives                        | Description                                                              |
|-----------------------|-----------------------------------------------|--------------------------------------------------------------------------|---------------------------------------------------------------------------|-------------------------------------|---------------------------------------------|--------------------------------------------------------------------------|
| Workload1<br>{sample} | Daily volume:<br>MB/TB<br><br>% annual growth | 24-hr period<br>8-hr workday<br>Batch Load<br>▶ Frequency<br>▶ Timeframe | Avg. TXN size<br>Total # of Tables<br>% Inserts<br>% updates<br>% deletes | Seconds<br>Minutes<br>Hours<br>Days | Milliseconds<br>Seconds<br>Minutes<br>Hours | Describe workload<br>Concurrency<br>Type of queries<br>Columns per Table |
| Workload2             |                                               |                                                                          |                                                                           |                                     |                                             |                                                                          |
| Workload3             |                                               |                                                                          |                                                                           |                                     |                                             |                                                                          |
| Workload4             |                                               |                                                                          |                                                                           |                                     |                                             |                                                                          |



# B

## Java API sample code snippet

In this appendix, we present Java API sample code snippets.

# Available metadata in the DVM server

This sample code snippet lists all available metadata in the DVM server.

This example programmer code that is shown in Figure B-1 is representative of the output for the included reusable code snippet.

```

Connecting using url
url = jdbc:rs:dv:DatabaseType=DVS; Host=9.212.143.72; Port=1200;
Connected !

Run 1
30.07.2020 15:58:52:0000539 Fetch Metadata - START
30.07.2020 15:58:52:0000539 Conn.getMetaData(): DvsDatabaseMetaData@671dae58 (0 ms)
databaseMetadata=DvsDatabaseMetaData@671dae58
catalog: null, schemaPattern: null, tableNamePattern: %, types: null
tables=DvCmbuResultSet@d3206849
columnCount=10

TABLE_CAT | TABLE_SCHEM | TABLE_NAME | TABLE_TYPE | REMARKS | TYPE_CAT | TYPE_SCHEM | TYPE_NAME | SELF_REFERENCING_COL_NAME | REF_GENERATION |

1 | DVSQ | STAFFVS | TABLE | VSAM - FBOR.DVM.V1R1.STAFF.VSAM | null | null | null | null | null |
2 | DVSQ | SYSLOGD | TABLE | zFS file - /tmp/syslogd.log | null | null | null | null | null |
30.07.2020 15:58:52:0000639 Processed DatabaseMetaData.getTables().getColumns(). Tables:2 Columns: 10 (100 ms)
30.07.2020 15:58:52:0000639 Fetch Metadata - END. START-to-END took total 100 ms (0 secs).

Closed !

```

Figure B-1 Sample output for the included reusable code snippet

The example output in Figure B-2 is representative of output that includes all available columns for two virtual tables over VSAM (STAFFVS) and zFS (SYSLOGD) data sources

```

Connecting using url
url = jdbc:rs:dv:DatabaseType=DVS; Host=9.212.143.72; Port=1200;
Connected !

Run 1
30.07.2020 16:12:22:0000634 Fetch Metadata - START
30.07.2020 16:12:22:0000634 Conn.getMetaData(): DvsDatabaseMetaData@dd5bb898 (0 ms)
databaseMetadata=DvsDatabaseMetaData@dd5bb898
catalog: null, schemaPattern: null, tableNamePattern: %, colNamePattern: null
30.07.2020 16:12:22:0000819 DatabaseMetaData.getTables(): DvCmbuResultSet@6e2fc0e9 (185 ms)
tables=DvCmbuResultSet@6e2fc0e9
columnCount=24

TABLE_CAT | TABLE_SCHEM | TABLE_NAME | COLUMN_NAME | DATA_TYPE | TYPE_NAME | COLUMN_SIZE | BUFFER_LENGTH | DECIMAL_DIGITS | NUM_PREC_RADIX | NULLABLE | REMARKS | CC

1 | DVSQ | STAFFVS | STAFFVS_KEY_ID | 5 | SMALLINT | 5 | 2 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM |
2 | DVSQ | STAFFVS | STAFFVS_DATA_NAME_L | 5 | SMALLINT | 5 | 2 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM |
3 | DVSQ | STAFFVS | STAFFVS_DATA_NAME | 1 | CHAR | 9 | 9 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM | 0 |
4 | DVSQ | STAFFVS | STAFFVS_DATA_DEPT | 5 | SMALLINT | 5 | 2 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM |
5 | DVSQ | STAFFVS | STAFFVS_DATA_JOB | 1 | CHAR | 5 | 5 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM | 0 |
6 | DVSQ | STAFFVS | STAFFVS_DATA_YRS | 5 | SMALLINT | 5 | 2 | 0 | 10 | 0 | VSAM - FBOR.DVM.V1R1.STAFF.VSAM |
7 | DVSQ | SYSLOGD | DATE | 1 | CHAR | 7 | 7 | 0 | 10 | 0 | zFS file - /tmp/syslogd.log | 0 | 0 | 0 | 1 |
8 | DVSQ | SYSLOGD | TIME | 1 | CHAR | 8 | 8 | 0 | 10 | 0 | zFS file - /tmp/syslogd.log | 0 | 0 | 0 | 2 |
9 | DVSQ | SYSLOGD | SYSID | 1 | CHAR | 4 | 4 | 0 | 10 | 0 | zFS file - /tmp/syslogd.log | 0 | 0 | 0 | 3 |
10 | DVSQ | SYSLOGD | MESSAGE | 12 | VARCHAR | 490 | 490 | 0 | 10 | 0 | zFS file - /tmp/syslogd.log | 0 | 0 | 0 |
30.07.2020 16:12:22:0000835 Processed DatabaseMetaData.getTables().getColumns(). Tables:10 Columns: 24 (16 ms)
30.07.2020 16:12:22:0000835 Fetch Metadata - END. START-to-END took total 201 ms (0 secs).

Closed !

```

Figure B-2 Sample output of Java code snippet with all columns

Example B-1 shows a reusable code snippet for listing metadata.

*Example B-1 Code for listing metadata*

---

```
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Properties;

public class DVMMetadataTablesApp {
 protected Properties props = new Properties();
 protected Connection conn;
 protected Statement stmt;
 protected ResultSet rs;
 private long start;
 private DateFormat df = new SimpleDateFormat("dd.MM.yyyy
HH:mm:ss:SSSSSS");
 private long previousTime = start;
 protected Connection getConnection()
throws Exception {
String url = "jdbc:rs:dv:DatabaseType=DVS; Host=XXX; Port=YYY;";
 props.setProperty("user", "AAA");
 props.setProperty("password", "BBB");

show("-----");
 show("Connecting using url ");
 show("url = " + url);
 conn = DriverManager.getConnection (url, props);
 show("Connected !");
 show("-----");
 conn.setAutoCommit(false);
 return conn;
}

public static void main(String[] args) throws Exception
{
DVMMetadataTablesApp dvmApp = new
DVMMetadataTablesApp();
dvmApp.getConnection();
 for(int i=1; i <= 1; i++) {
 dvmApp.show("Run " + i);
 dvmApp.fetchMetadata();
 dvmApp.show("");
 }

 dvmApp.closeAll();
}

private void show(Object msg)
{
 System.out.println(msg);
}

private void fetchMetadata() throws Exception
{
 // TODO Auto-generated method stub
 start = System.currentTimeMillis();
}
```

```

 log("Fetch Metadata - START", false);
 DatabaseMetaData databaseMetadata = conn.getMetaData();
 log("Conn.getMetaData(): " + databaseMetadata, true);
 printWithCRLF("databaseMetadata=" + databaseMetadata);
 String catalog = conn.getCatalog();
 String schemaPattern = null;
 String tableNamePattern = "%";
 String[] types = null;
 printWithCRLF("catalog: " + catalog + ", schemaPattern: " + schemaPattern + ",
 tableNamePattern: " + tableNamePattern + ", types: " + types);
 ResultSet tables = databaseMetadata.getTables(catalog, schemaPattern,
 tableNamePattern, types);
 printWithCRLF("tables=" + tables);
 int columnCount = printHeader(tables);
 int rowCounter = 0;
 while(tables.next())
 {
 for(int i=1; i <= columnCount; i++)
 {
 String value = tables.getString(i);
 if(value == null && i == 1)
 {
 value = " " + ++rowCounter + " ";
 }
 printWithCRLF(value + " |\\t");
 }
 printWithCRLF("");
 }
 log("Processed DatabaseMetadata.getTables().getColumns(). Tables:" + rowCounter +
 " Columns: " + columnCount, true);
 long end = System.currentTimeMillis();
 long elapsed = end - start;
 log("Fetch Metadata - END. START-to-END took total " + elapsed + " ms (" +
 (elapsed/1000) + " secs).", false);
 closeStatement();
}
private int printHeader(ResultSet tables) throws SQLException
{
 ResultSetMetaData resultSetMetadata = tables.getMetaData();
 int columnCount = resultSetMetadata.getColumnCount();
 printWithCRLF("columnCount=" + columnCount);
 printWithCRLF("-----");
 int lineLength = 0;
 for(int i=1; i <= columnCount; i++) {
 printWithCRLF(resultSetMetadata.getColumnName(i) + " | ");
 lineLength += resultSetMetadata.getColumnDisplaySize(i);
 }
 printWithCRLF("");
 for(int i=1; i <= lineLength; i++)
 {
 printWithCRLF("-");
 }
 printWithCRLF("");
 return columnCount;
}

```



```
private void printWithCRLF(String message)
{
 System.out.println(message);
}

private void printWithOUTCRLF(String message)
{
 System.out.print(message);
}
```

---





# Troubleshooting and diagnosing

This appendix focuses on best practices for troubleshooting and diagnosis of general problems that are encountered with the IBM Data Virtualization Manager for z/OS (DVM for z/OS) technology.

When you encounter problems with your environment or application, it is always good to know where to start and how to characterize the situation. This appendix focuses on initial assessment, must gather, and server trace practices in displaying, viewing, and gathering details.

If IBM Support is required, some direction also is provided specifically to “must gather” materials.

For specific technical problems or an inability to procure PTFs or updated client components, open a problem ticket with IBM Support.

This appendix includes the following topics:

- ▶ “Initially characterizing problems” on page 240
- ▶ “Must Gather information” on page 240
- ▶ “Capturing trace browse from the DVM server ISPF panel” on page 241
- ▶ “Search Techdocs for answers” on page 248

## Initially characterizing problems

Suitable context around the environment, configuration, accessing applications, and active workloads are critical to determining the severity of a problem and tracking down its resolution.

The following questions are useful to assess the problem and overall context:

- ▶ Is this a new or existing application?
- ▶ Has anything changed in the environment, application, or configuration?
- ▶ If failure occurred, what is the display or recorded message?
- ▶ Did the problem result in a down or unproductive system?
- ▶ Did the problem occur in a Test, Development, QA, or Production environment?
- ▶ Is the problem encountered delaying a production deployment?
- ▶ What is the timeframe for a resolution to your problem?
- ▶ Can you reproduce the problem and capture output to support your experience?
- ▶ If your problem can be reproduced, can you provide the exact steps for reproduction?
- ▶ Can sample SQL or source code that affects the problem be provided?

## Must Gather information

If you are unsuccessful in [searching available Techdocs](#) for your problem, you can gather more information about your environment, application, and the behavior that was experienced and start a problem ticket with IBM Support.

## DVM for z/OS version

IBM distributes and supports version 1.1.0 of DVM for z/OS.

## High Module date

Provide a complete copy of your DVM server JOBLOG. This file contains the following information and more, which helps the IBM support team. The job log is a record of job-related information for the programmer, as shown in the following example:

```
AVZ0340I SVFXLevel = 2016/02/18 13:40:50 SVFX0000 01.01.00
AVZ0340I High module OPTPIN assembled at: 2016/02/18 13.40
```

## PTF Level

The PTF level can be found that uses the sample job that is shown in Example C-1.

*Example C-1 Sample job to get PTF level*

---

```
//GIMSMP EXEC PGM=GIMSMP,REGION=0M
//SMPCSI DD DISP=SHR,DSN=<HLQSMP>.GLOBAL.CSI
//SMPLOG DD DUMMY
//SMPOUT DD SYSOUT=*
//SMPCNTL DD *
 SET BDY(<TGTZONE>) .
 LIST SYSMODS XREF .
/*
```

---

## Operating system

Provide the following information about the operating system:

- ▶ Level of z/OS in use
- ▶ IBM Z architecture that is in use (for example, z13, z14, or z15)

## DVM server environment

Provide the following information about the DVM server environment:

- ▶ Client operating system where DVM Studio is installed
- ▶ Version of Microsoft Windows and whether the workstation is 32-bit or 64-bit
- ▶ Build version of DVM Studio
- ▶ Version of JDBC or ODBC driver in use

## Capturing trace browse from the DVM server ISPF panel

Trace Browse is a facility in which the DVM server logs critical events that can be viewed to help diagnose, debug, and correct problems that are encountered with the configuration or execution of the solution.

The trace adds records to a trace buffer that is maintained in virtual storage. When a session completes, the trace records are automatically saved in a VSAM data set.

## Capturing a copy of the Trace Browse

DVM for z/OS has one principle mechanism for diagnostics and troubleshooting. During the initial configuration of the DVM server, the system administrator is prompted to designate a data set for traces. The traces can be viewed from the DVM server ISPF panel or the desktop DVM Studio.

The trace is accessed from menu selection **B** in the ISPF interface, as shown in Figure C-1.

```
IBM Data Virtualization Manager for z/OS
Option ===> B

Interface Facilities:
1 ACI 5 IDMS SSID :
2 Adabas 6 IMS Version :
3 CICS 7 VSAM/Sequential Date :
4 DB2 8 DSSPUFI Time :

Data Virtualization Server Administration:
A Remote User - Manage Remote Users
B Server Trace - Server Trace Facility - SIS SSID: AVZW
```

Figure C-1 Selecting Server Trace from the DVM server ISPF panel

With the cursor in the Option prompt, press **F1** to see an inline tutorial for Server traces, as shown in Figure C-2.

```
Tutorial ----- Server Trace ----- Tutor
Option ==> █

Server Trace lets you view the list of product events captured by the
Server(s) executing in your environment. This list contains all of
the events (SQL, IMS, TCP/IP, LU 6.2, etc.) that have occurred.

- At your option, you may display formatted columns of information such as
userid, and time.
- You may use FIND and LOCATE commands to search for data or a specific
time and date.
- You may use the DISPLAY command to display additional columns of
information.
- You may use the STATUS command to display the Server Trace status area.

The following topics are presented in sequence or may be selected by
number.

1 The Server Trace Display - Moving about inside Server Trace
2 Primary Commands - Using Server Trace
3 Line Commands (LCs) - Line commands allowed in Server Trace
```

Figure C-2 Server Trace inline tutorial

Paging through the tutorial displays DISPLAY, LOCATE, FIND, RFIND, and PROFILE. Double-clicking any of these words ZOOMS into details for the command. For example, select **DISPLAY** to see more information about that command, as shown in Figure C-3.

```
Option ==>

The DISPLAY command is used to control the display of formatted columns of
message information. D can be used as the abbreviation for DISPLAY. When
visible, the columns of information appear to the left of the message texts
and are not scrolled offscreen when the LEFT and RIGHT scroll commands are
used. You may display as many as 5 columns at one time. Example:

 DISPLAY DATE TIME USERID ASID SECONDS

Column Abbr Description
----- -
ACTION AC displays the final SEF event action, accept (ACC),
 reject (REJ), or noaction (NOA)
ADDRESS AD location in memory of the message data
ADDRJOB ADDRJ location in memory of entry in jobname vector
ADDRUSR ADDRU location in memory of entry in userid vector
APMRC APM APPC/MVS return code

More: +
```

Figure C-3 ZOOM on display

## Displaying and viewing server traces

Server trace information can be viewed and printed from the ISPF interface and DVM Studio.

### Using the ISPF panel to display and view the server trace

Entering 1 at the command prompt in the Server Trace ISPF panel displays the latest Server Trace that was written, as shown in Figure C-4.

```
----- Server Trace --- 10:41:45 08 MAY 20 Co1s 001 060
Command ==>
HH:MM:SS HOST NAME -----1-----2-----3-----4-----5-----6
10:41:45 ACI 00007719 0017B015 TIMED OUT
10:41:45 ACI 00007719 0017B015 DEREGISTERED SERVER SET TO TIMEOUT ST
10:41:45 ACI 00007720 0017B015 TIMED OUT
10:41:45 ACI 00007720 0017B015 DEREGISTERED SERVER SET TO TIMEOUT ST
```

Figure C-4 Server trace display

Complete the following steps:

1. Connect to the DVM ISPF panel.
2. Select **B** (Server Trace).
3. On the command line, run the following command and press **Enter**:  
D MSGNO T CNID
4. Print a block of the trace by entering **SS** over the **MSGNO** column from the lines to copy from and **SS** over the **MSGNO** column for where to copy to, and then, press **Enter**.  
The printed block of the trace is written to the ISPF list data set.
5. Run the **LIST** command from the command line and specify option 3 to keep the listed data set and allocate a new one.
6. Terse the trace data set and attach it to your problem ticket.

The display shows the timestamp for the trace record without the date the trace record was written. Running the following command at a command-line prompt displays date and time in DDMMM HH:MM:SS format, as shown in Figure C-5:

D DATE TIME

```
Command ==>
DDMMM HH:MM:SS -----1-----2-----+
08MAY 15:27:42 ACI 00000003 0017B03B SU
08MAY 15:27:42 ACI 00008118 0017B03B SU
08MAY 15:27:42 ACI 00000004 0017B036 RE
08MAY 15:27:42 ACI 00000004 0017B036 SU
08MAY 15:27:42 ACI 00008119 0017B036 SU
08MAY 15:27:42 ACI 00008120 0017B033 SU
```

Figure C-5 Server trace display with date and time

This panel in the Server Trace Facility shows summary steps that are being traced. Double-clicking any line in this view ZOOMS into the capture for the specific record. The example in Figure C-6 shows processing an SQL statement with associated detail information.

```

----- Server Trace --- 15:40:47 08 MAY 20 Cols 001 064
Command ==> Scroll ==> CSR
DDMMM HH:MM:SS ---+---1---+---2---+---3---+---4---+---5---+---6---
08MAY 15:40:47 WRITE EXECUTED - SOCK 0002 - WRITE COMPLETED
08MAY 15:40:47 READ EXECUTED - SOCK 0002 - READ COMPLETED
08MAY 15:40:47 SQL engine opened, client CCSID = 37, result set CCSID = 1047
08MAY 15:40:47 SQL ENGINE OPEN DATABASE - RC 0
08MAY 15:40:47 SET AUTO-OFF - SQLCODE 0
08MAY 15:40:47 WRITE EXECUTED - SOCK 0002 - WRITE COMPLETED
08MAY 15:40:47 READ EXECUTED - SOCK 0002 - READ COMPLETED
08MAY 15:40:47 SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HI
08MAY 15:40:47 SQL ENGINE HPO OPEN-CURSOR - SQLCODE 0

VIEW User Trace Control Block Columns 00001 00072
Command ==> Scroll ==> PAGE
000178 +00A0 40404040 00D60000 000400D6 00000002 * .0.....0... * *@@@.0...
000179 +00B0 00000000 00000000 00000000 00000000 * * *
000180 +00C0 To 010F Same As Above
000181 +0110 00000000 00000000 00000000 0000FFFF * * *
000182 +0120 0A11310F C3D4C9C4 00000000 00000000 * ...CMID..... * *.1.CMID.
000183 +0130 FFFF0002 7DB12008 00000000 00000000 * * *
000184 +0140 0000FFFF 0A11310F C2D68100 F230FAFC *BOa.2... * *.1.B
000185 +0150 1D9DEDE8 0089D7A0 00000000 012110AD * ...Y.iP..... * *.Y.iP..
000186 +0160 00000000 0B272ABE 00000000 0593FCD6 *l.0* * *
000187 +0170 0000005E 0000005E 00000048 00000000 * ...;.;..... * *.~.~.
000188 +0180 00000000 00000000 00000000 00000000 * * *
000189 +0190 03010778 07E30A17 00000000 00000000 *T..... * *.X.T..
000190
000191 SQLCA Control Block
000192

```

Figure C-6 Server Trace display with detail for SQL statement

This type of Server Trace information is helpful when working with IBM Support. At times, the system administrator might need to capture a Server Trace for a specific situation that occurred or can be re-created in a separate environment and send it for further analysis.

## Capturing and printing server trace output

To capture and print the server trace output, the administrator must export the trace to a data set by completing the following steps:

1. From the Server Trace Command prompt, enter the following command and press **Enter**:  
D MSGNO T TCB
2. Enter PP over the displayed MSGNO column for the start line of the block to copy and PP over the MSGNO column for the last line of the block to copy.

**Tip:** Capturing more lines before and after the problem error as a good practice. Substitute SS for PP for zoomed trace to get all of the underlying content you get by double-clicking.

3. Run the **LIST** command from the Server Trace command prompt.
4. Specify option 3 to keep the list data set and allocate a new one.
5. Download the list data set and attach the file to your IBM Support problem ticket.



## Using DVM Studio to display and view a server trace

Chapter 7, “Managing and monitoring” on page 131 describes how to use DVM Studio to display, view, capture, and export Server Trace information.

## Using DVM Studio to diagnose SQL results

A more accessible form of debugging is in the SQL Results tab and the Console View. This example includes a stray key stroke in the SQL that makes the virtual table name invalid, as shown in the SELECT statement in Figure C-7. This issue results in the SQL error attempting to run the statement.

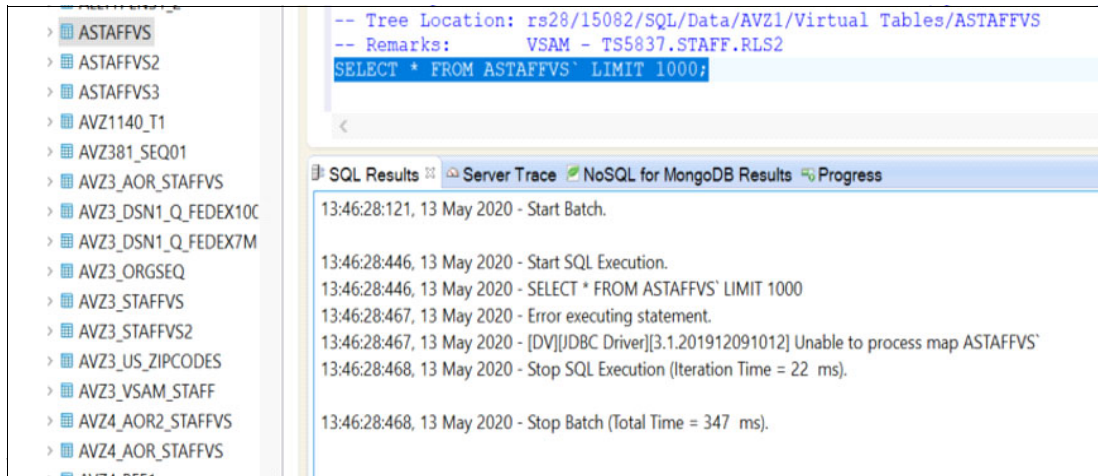


Figure C-7 SQL results error message from the SQL tab

The SQL results show the error Unable to process map ASTAFFVS. Also of interest to IBM Support and developers is the version number of the JDBC driver:

```
[DV][JDBC Driver][3.1.201912091012]
```

The console provides more detailed messaging, as shown in Figure C-8.

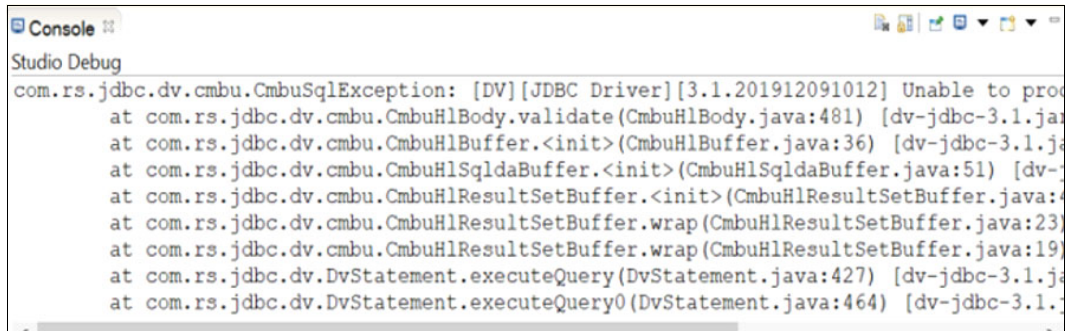


Figure C-8 Detailed message from the Console

In the event, the Console View is closed. Click **Show View** → **Other**. Entering the first few letters of Console displays the icon as a selection for the console that can be selected to continue diagnosis, as shown in Figure C-9.

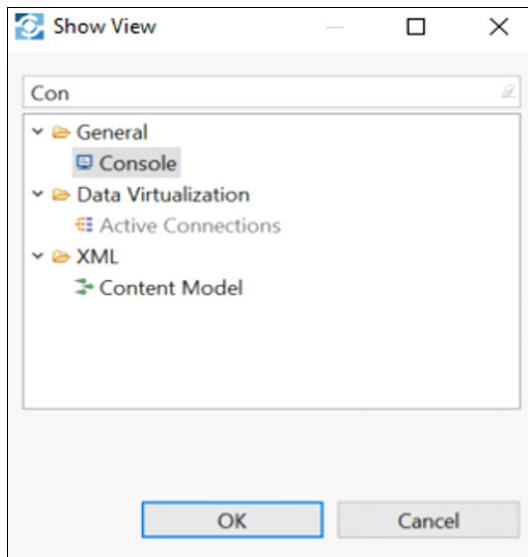


Figure C-9 Show View input field

## Consolidating server trace over multiple DVM servers

This section focuses primarily on a single DVM server subsystem for working with Server Trace. However, when multiple DVM servers are participating in a single LPAR, across LPARs, or part of a Sysplex, the settings can be adjusted in the IN00. This feature helps the administrator centralize all DVM subsystem server traces to a single repository so that analysis and troubleshooting can be performed more easily.

This process is done by enabling the DVM Instrumentation Server in the IN00 file, as shown in Figure C-10.

```

058200 /* -----
058300 /* Enable the Instrumentation Server (SIS).
058400 /* SIS/XCF - Enable this Server address space to route
058500 /* - Server Trace data to SIS.
058600 /* CONNID - Any meaningful 8 character name.
058700 /* SISID - 4 char. name that must match the name in
058800 /* - the AVZSIN00 (SIS address space startup)
058900 /* - parameters.
059000 /* SISXCFGRP - 4 char. name that must match the name in
059100 /* - the AVZSIN00 (SIS address space startup)
059200 /* - parameters.
059300 /* MANAGER - YES/NO. Indicates that this Server address
059400 /* - space is a SIS manager or not.
059500 /*
059600 /* There can be multiple DEFINE statement groups which enables you
059700 /* to route a single Server address space's messages to multiple
059800 /* SIS'.
059900 /*

```

Figure C-10 Edit the IN00 file to enabled the DVM Instrumentation Server (SIS)

Customize the server trace to enable the Instrumentation Server (SIS) by defining the parameters from Figure C-11 for the JCL shown in Figure C-10 on page 246.

```

Command ==> SCOT
060000 if DontDoThis then
060100 do
060200 "MODIFY PARM NAME(SIS/XCF) VALUE(YES)"
060300
060400 "DEFINE SISXCF",
060500 "CONNID(SYS1AVZW)",
060600 "SISID(AVZ1)",
060700 "SISXCFGRP(SISAVZ1)",
060800 "ERRORONLY(YES)",
060900 "MANAGER(NO)"
061000 end
061100

```

Figure C-11 Add code to define the SISXCF server

Figure C-12 shows an example of combining the logs from two different LPARs. The fields inside the IF statements are used to identify which records come from each LPAR.

```

CSD.AI38.EXECFB(AVZMIN00) - 01.33 Column:
do
"MODIFY PARM NAME(SIS/XCF) VALUE(YES)"

"DEFINE SISXCF",
"CONNID(JEFF)",
"SISID(BERT)",
"SISXCFGRP(JEAVZ1)",
"ERRORONLY(NO)",
"MANAGER(YES)"

"DEFINE SISXCF",
"CONNID(GREGG)",
"SISID(ERNI)",
"SISXCFGRP(GRAVZ1)",
"ERRORONLY(NO)",
"MANAGER(YES)"
end

```

Figure C-12 Combined Server Trace logs from two DVM servers over two LPARs

Figure C-13 shows combined Server Trace output for both DVM servers. RS28AVZ1BERT is one LPAR, RS22AVZ1ERNIE is a second LPAR, and SISLOCALRS28 is the local DVM server for the present environment.

```

MSGORIGIN/SISXCF HH:MM:SS DDMM ---+---1---+---2---+---3---
RS28AVZ1BERT 16:20:12 21MAY AV12778T Zconnect Server AVZ1N1
RS22AVZ1ERNI 16:20:18 21MAY AV12711T Cannot start Zconnect S
RS22AVZ1ERNI 16:20:18 21MAY AV12778T Zconnect Server AVZ1N1
RS22AVZ1ERNI 16:20:18 21MAY AV12778T Zconnect Server AVZ1N1
RS22AVZ1ERNI 16:20:18 21MAY AV12778T Zconnect Server AVZ1N1
RS22AVZ1ERNI 16:20:18 21MAY AV12778T Zconnect Server AVZ1N1
RS22AVZ1ERNI 16:20:19 21MAY RESMGR detected termination of m
SISLOCALRS28 16:20:22 21MAY AVM2711T Cannot start Zconnect S
SISLOCALRS28 16:20:22 21MAY AVM2778T Zconnect Server AVZMN1
SISLOCALRS28 16:20:22 21MAY AVM2778T Zconnect Server AVZMN1
SISLOCALRS28 16:20:22 21MAY AVM2778T Zconnect Server AVZMN1
RS28AVZ1BERT 16:20:22 21MAY AV12711T Cannot start Zconnect S
RS28AVZ1BERT 16:20:22 21MAY AV12778T Zconnect Server AVZ1N1
RS28AVZ1BERT 16:20:22 21MAY AV12778T Zconnect Server AVZ1N1
RS28AVZ1BERT 16:20:22 21MAY AV12778T Zconnect Server AVZ1N1
RS28AVZ1BERT 16:20:22 21MAY AV12778T Zconnect Server AVZ1N1

```

Figure C-13 Server Trace output for both DVM servers

## Search Techdocs for answers

Answers to many problems can be found on the [IBM Support website](#).

Entering keywords into the search field always is the first recommendation when you encounter a problem in your running environment.

Over 150 Techdocs about DVM for z/OS are available that can assist in solving your technical issue before an IBM Support problem ticket is opened, as shown in Figure C-14.

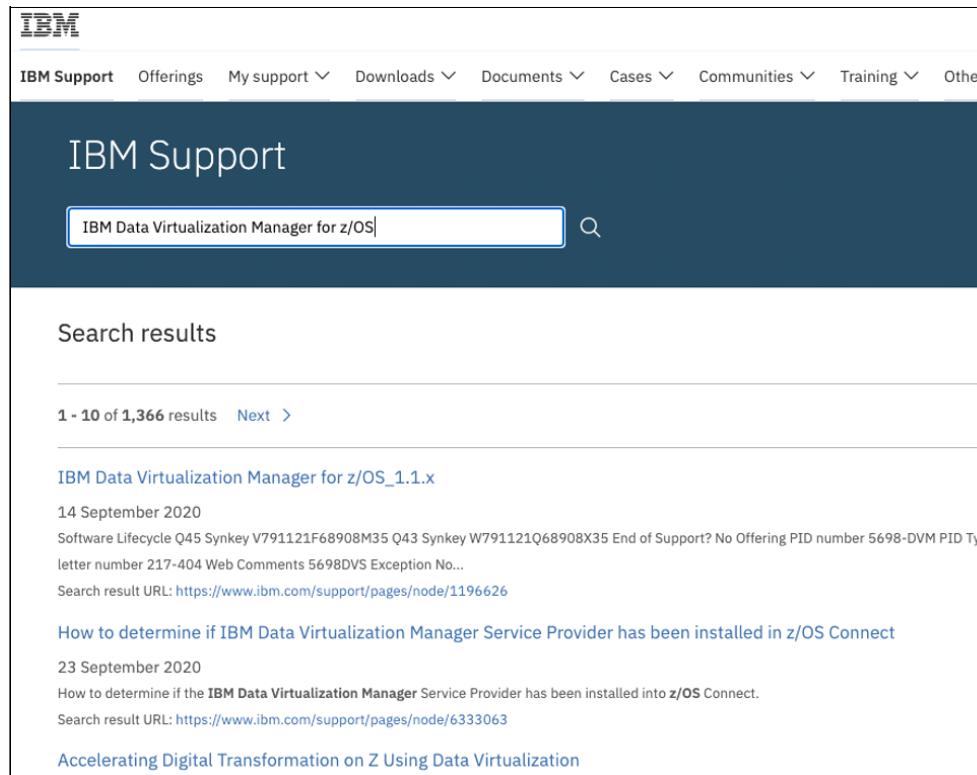


Figure C-14 IBM Support website

Table 1 lists frequently encountered problems when first starting to work with DVM for z/OS. This shortlist represents topics that are available by searching the web. Links to these Technotes can be found in the [Troubleshooting and Diagnosis Technote](#).

Table C-1 Popular DVM for z/OS Technotes

| Category            | Technotes                                                                                                                                                                                                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Data                | How to control the number of rows returned on a SQL query using the Data Virtualization Manager ODBC driver?                                                                                                                                                                                                                   |
| SSL                 | <ul style="list-style-type: none"> <li>▶ How to resolve the "Communication link broken" and "Invalid HTTP headers - NETWORK I/O ERROR" error when setting up server only SSL with Data Virtualization ODBC Drivers</li> <li>▶ Setting up Driver when using an SSL certificate to access Data Virtualization Manager</li> </ul> |
| JDBC Gateway server | <ul style="list-style-type: none"> <li>▶ JGATE running on UNIX System Services gets "Undefined Error" when connecting to Oracle</li> <li>▶ How to launch JGate as a started task?</li> <li>▶ Change JGATE Memory Size on Startup</li> <li>▶ How to connect from JGate to other databases using TLS1.2</li> </ul>               |

| Category                              | Technotes                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Performance                           | Db2 Performance in DVM                                                                                                                                                                                                                                                                                                                                                   |
| Load balance configuration            | <ul style="list-style-type: none"> <li>▶ Load Balance and Failover of CICS Regions</li> <li>▶ How to configure TCP/IP shareportwlm for Server load balancing</li> </ul>                                                                                                                                                                                                  |
| DVM Studio                            | <ul style="list-style-type: none"> <li>▶ Failure to export configuration</li> </ul>                                                                                                                                                                                                                                                                                      |
| Server trace                          | <ul style="list-style-type: none"> <li>▶ How to eliminate XTX messages to be written to Server Trace browse</li> <li>▶ How to send messages from a Natural program to Trace Browse using ACI API</li> </ul>                                                                                                                                                              |
| z/OS Connect Service Provider logging | z/OS Connect Service Provider Logging Control                                                                                                                                                                                                                                                                                                                            |
| Service ISPF activity display         | How to Enable/Populate "Interval Activity" (Option F.1) in Data Virtualization Server ISPF panel                                                                                                                                                                                                                                                                         |
| Others                                | <ul style="list-style-type: none"> <li>▶ Unable to connect to the DVM server from DVM Studio</li> <li>▶ How to connect the JDBC Gateway to relational databases</li> <li>▶ Encountering a syntax error when creating a Virtual Table (sample program to resolve)</li> <li>▶ Hints on determining whether the DVM server is configured for z/OS (most popular)</li> </ul> |

Also, DVM for z/OS includes a new IBM Community that is available to submit questions, download sample videos, and so on. IBM also produces IBM demonstrations that can help introduce new users to the product through demos, product tours, and hands-on labs:

- ▶ [IBM Data Virtualization Manager for z/OS Community](#)

A valid IBM user ID and password is required to log in to the IBM DVM Community for access to various resources, white papers, videos, and so on, to learn more about different uses of the technology and interact with other community members.

- ▶ [IBM Demos for IBM Data Virtualization Manager for z/OS](#)



**Redbooks**

**IBM Data Virtualization Manager for z/OS**

(0.2"spine)  
0.17"->0.473"  
90->249 pages









SG24-8514-00

ISBN 0738459984

Printed in U.S.A.

Get connected

