DB2 11 for z/OS

An IDUG User Technical Perspective

Abstract

In March 2013 the IDUG DB2 11 Editorial Committee was formed, comprised of volunteers from IDUG's worldwide community of DB2 users and consultants. Working alongside IBM's formal Early Support Program (ESP), the Editorial Committee gained valuable insight into exactly what makes the new release tick. This White Paper contains their findings, including practical experiences and independent evaluations of the new features.

IDUG DB2 11 Editorial Committee

October 2013

TABLE OF CONTENTS

INTRODUCTION

ABOUT IDUG

The International DB2 Users Group (IDUG®) is an independent, not-for-profit, user-run organization whose mission is to support and strengthen the information services community by providing the highest quality education and services designed to promote the effective utilization of DB2.

More information can be found on the IDUG website: www.idug.org

INTRODUCING THE IDUG DB2 11 EDITORIAL COMMITTEE

The Editorial Committee that produced this white paper was comprised of volunteers from IDUG's worldwide community of DB2 users and consultants.

Ron Brown

Ron Brown has been working with DB2 as a systems programmer and DBA for over 20 years. He is currently a member of the IDUG Australasia conference planning committee.

Cuneyt Goksu

Cuneyt Goksu is Principal Information Management Consultant in VBT, specialising in DB2 consultancy, education and software development. He studied Computer Science, holding MBA and MS in CS specialized in Database Modeling. He has been working with DB2 more than 20 years.

Cuneyt is an IBM DB2 Gold Consultant, Member of Board of Directors for the International DB2 User Group and an IBM Champion. He can be contacted at CuneytGoksu@Usa.Net

Dan Luksetich

Dan Luksetich is a data scientist and senior DB2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 28 years, and has worked with DB2 for over 23 years. He has been a COBOL and BAL programmer, DB2 system programmer, DB2 DBA, and DB2 application architect. His experience includes major implementations on z/OS, AIX, i Series, Windows, and Linux environments.

Dan works everyday on some of the largest and most complex DB2 implementations in the world. He is a certified DB2 DBA for DB2 for z/OS and LUW, system administrator for z/OS, and application developer, and has worked on the teams that have developed the DB2 for z/OS certification exams. He is the author of several DB2 related articles as well as co-author of the DB2 9 for z/OS Certification Guide and the DB2 10 for z/OS Certification Guide.

Bjarne Nelson

Bjarne Nelson has worked full time since 1985 with most flavors of DB2, especially focusing on performance, distribution, and large scale data warehousing, geographically covering mainly Europe, but other continents as well. Since 1993 he has served in numerous positions as an IDUG volunteer and is now on the Board of Directors as President Elect. He participates in IBM's Gold Consultants Program as well as being an IBM Champion for Data Management. Nelson has often presented at conferences like IDUG, CMG and Xephon Briefings.

Achievements as DB2 consultant include implementing large scale DB2 data warehousing, playing a key role in major migrations to DB2, involvement in world class Data Sharing implementations, and engaging as mainframe adviser in web based development projects.

Toine Michielse

Toine Michielse is the lead architect for DB2 on z/OS at SwissRe, Switzerland. He has over 25 years of experience with DB2 as a programmer, DBA, system programmer, teacher and consultant. In the past, Toine has also been working for DB2 development as a lab advocate.

Toine can be reached at: toine_michelse@swissre.com

Cristian Molaro

Cristian Molaro is an independent DB2 specialist and an IBM Gold Consultant. His main activity is linked to DB2 for z/OS administration and performance. He is co-author of 8 IBM Redbooks related to DB2 including the recent "DB2 11 for z/OS Technical Overview". He obtained the merit badge “Platinum IBM Redbook Author”.

He has presented papers at several international conferences and local user groups in Europe and North America and was recognized by IBM as an Information Champion in 2009, 2010, 2011, 2012 and 2013. Cristian is part of the IDUG EMEA Conference Planning Committee, and he is Chairman of the DB2 LUW User Group BeLux.

He was recognized by IBM as "TOP" EMEA Consultant at IDUG EMEA DB2 Tech Conference Prague 2011. Cristian can be reached at cristian@molaro.be or at be.linkedin.com/in/cristianmolaro/.

Kurt Struyf

After spending part of his teenage years in Texas, Kurt Struyf started his career at a major Belgian bank as part of the DB2 team. He continued his path with an outsourcing company providing technical support to their range of customers, offering design advice and installing and maintaining their DB2s on z/OS.

Now Kurt is working for SuadaSoft in Luxemburg as an independent consultant with customers all over Europe. He has been involved in the implementation of complex database designs and high performance applications. Kurt works all over Europe and the USA, performing installations, migrations, tuning and other (system) DBA jobs. Besides his consultancy work, Kurt also teaches many DB2 and related topics for IBM and others throughout Europe and the USA. He is a recognized speaker and has been a speaker at several IDUG conferences and regional user groups as well as IOD conferences both in Europe and the USA. Kurt in an “IBM information Champion”.

Julian Stuhler

Julian Stuhler is a Principal Consultant with Triton Consulting, a UK-based company specialising in the provision of DB2 consultancy, education, software and managed services to clients throughout Europe. He has over 25 years relational database experience as a DBA and Systems Programmer.

Julian is an IBM DB2 Gold Consultant, Past-President of the Board of Directors for the International DB2 User Group and an IBM Champion. He can be contacted at julian.stuhler@triton.co.uk.

Billy Sundarrajan

Thanikachalam "Billy" Sundarrajan has over 23 years of experience in the design, administration, and performance tuning of large DB2 z/OS based database systems. He currently manages the database team at Fifth Third Bank and is responsible for the design, maintenance, and performance tuning of database systems across multiple lines of businesses.

Billy is an IBM Information Champion, and an IDUG volunteer serving in the IDUG Board of Directors. Billy has presented on wide and varying database topics at IOD, IDUG North America, IDUG EMEA and multiple DB2 user groups. Previously, he worked as a manager for Deloitte Consulting with a primary focus on implementation and performance tuning of large DB2-based state public welfare computer systems.

Isaac Yassin

Isaac Yassin is an IBM Gold Consultant & IBM Champion – Information Management whose experience in computer systems dates back to 1976. Isaac works mainly as a consultant to many organizations, both in Israel and Europe, covering wide aspects of performance tuning, troubleshooting, system design and education. He's been working closely with DB2 since its debut and followed up all versions and is an IBM certified solution expert and an IBM Certified Database Administrator for DB2 V7 / 8 / 9 / 10.

Isaac is a well-known lecturer and conducts many courses dealing with computer science (mainly in the DBMS area). Isaac holds a bachelor degree in Economics and Computer Science from BAR-ILAN University and holds several other titles from MAMRAM computer center at the IDF. Isaac serves as Israel RUG co-leader and also served on IDUG EMEA CPC 2005-2008 & 2010. Isaac serves now on the IDUG GMC.

HOW THIS PAPER IS ORGANISED

DB2 11 contains a large number of enhancements, but some will be of more interest than others depending on your job role and background. We have organised this paper according to the features likely to be most applicable to each of the three major technical roles in a DB2 for z/OS environment: Systems Programmers, Database Administrators and Application Developers.

However, many features are of interest to more than one role. Therefore, each major chapter ends with a “See Also” section cross-referencing other features which may also be of interest.

SYSTEMS PROGRAMMER TOPICS

This section of the document discusses new DB2 11 features that are expected to be of most interest to DB2 Systems Programmers.

EXTENDED LOG RECORD ADDRESSING

DB2 logging has been using a 6 byte RBA (Relative Byte Address) & LRSN (Log Record Sequence Number) for many years. The value continually increases until reaching end of range depending on amount of log records written for RBA or in year 2042 for LRSN (which is the high order 6 bytes from the 8 byte TOD clock). When that occurs - resetting the RBA back to zero requires a major outage of the DB2 subsystem, and there is no solution for end of LRSN values. Additionally, the LRSN value has a precision of only 16 microseconds which can lead to “LRSN spinning” in some intensive update situations, degrading DB2 performance.

DB2 11 resolves all of that by using 10 byte RBA or LRSN values. RBA gets extra high order bytes, increasing the maximum value to 2**80. LRSN gets 1 byte on the left and 3 on the right, to last for over 30,000 years and 16 million times more time precision. That change has far-reaching impact on DB2 – the BSDSs must be converted by an offline utility with DB2 down, and the page format of all other objects is updated by the normal REORG utility.

When you get to NFM mode you can convert the BSDSs, and that can be done one member at a time in a data sharing group to maintain continuous availability. Also in NFM mode you can change any of the tablespaces or indexes to the extended format pages whenever it suits; DB2 handles all objects in either 6 or 10 byte form.

IBM have done it well, allowing great flexibility for you to gradually convert everything to the extended RBA/LRSN. REORG/REBUILD utilities can even convert extended 10 byte page format back to basic 6 byte page format if desired. It is worth converting to the extended format because DB2 does everything internally using the 10 byte form and suffers a small overhead whenever it has to externalise a 6 byte value. IBM estimates that 6 byte logs add about 1% overhead, and converting logs to 10 byte and all tablespaces/indexes to extended format could give about 3% CPU saving.

The only obvious problem we found was related to the different log records – currently no replication software understands the extended records; IFCID 306 and the log capture exit have changed. Also, DSN1COPY copies of extended page format objects cannot be used by DB2 9 or 10, which only understand the basic (6 byte) page format. The extended format should allow DB2 systems to run for many years without having either of the two problems which triggered this change.

DRDA/DDF ENHANCEMENTS

DRDA usage is going up as people realize the benefits of using DB2 for z/OS as the main database behind their distributed working platforms. The appetite for more comprehensive support has driven up the demand for "bigger and better" stuff – the demand for longer Client User ID, Client Application Name, Client Accounting Information, Client Workstation Name and being able to have the Client Correlation Token has come from the need to have better compatibility within DB2 family and moreover. Another big request is the famous "can we do it faster?" – Response time continues to be a main issue and driving force behind DRDA usage (and where not?) – This request has been fulfilled in DB2 11 as well as some other requests.

CONNECTIVITY REQUIREMENTS

DB2 11 for z/OS communicates with distributed application using DRDA, the Distributed Relational Database Architecture protocol. When a communication is established, the client and the server defined the DRDA level to be used in the communication. The DRDA level defines which functionalities are going to be available during the communication, depending on the client and server capabilities.

The IBM Data Server Driver Package is recommended and simplifies application deployment using a registered DB2 Connect license. DB2 for z/OS V10 should operate with existing versions of DB2 Connect in place, even back to DB2 Connect V8. DB2 for z/OS will look into any connectivity related issues with existing apps using older versions of DB2 Connect. If any issues cannot be resolved within the DB2 for z/OS server, DB2 Connect will have to be upgraded to an in-service level which is currently V9.5.

DB2 Clients, Drivers and DB2 Connect DB2 10.5 FP2 are required to fully exploit DB2 11 for z/OS distributed features, such as:

- CALL with array type arguments
- Larger CLIENT INFO properties including the new Correlation Token field
- Implicit COMMIT for stored procedures
- Sysplex support for Global Session Variables

IBM Data Studio 4.1 provides support for DB2 11 for z/OS.

The usefulness of Implicit Commit is best shown by real-life numbers:

Delta(%) improvement from V10	Using Implicit Commit	
Workload	CL1 ET	Total CPU Times/Commit
Cobol Sproc	-23.19%	-4.81%
Native SQLPL	-17.90%	-5.14%
Java Sproc	-8.03%	-3.65%

CLIENT INFO ENHANCEMENTS

The DB2 drivers and clients support ***client info properties*** which you can use to provide extra information about the client to the server. This information can be used for accounting, workload management, resource control, or problem determination. Extended client information is sent to the database server when the application performs an action that accesses DB2 for z/OS, such as executing SQL.

The client info properties are:

- **Application Name**: The name of the application that is currently using the connection. This value is stored in DB2 special register CURRENT CLIENT_APPLNAME
- **Client Accounting Information**: The value of the accounting string from the client information that is specified for the connection. This value is stored in DB2 special register CURRENT CLIENT_ACCTNG
- **Client Workstation Name**: The host name of the computer on which the application that is using the connection is running. This value is stored in the DB2 special register CURRENT CLIENT_WRKSTNNAME
- **Client User**: The name of the user on whose behalf the application that is using the connection is running. This value is stored in the DB2 special register CURRENT CLIENT_USERID
- **Client Correlation Token**: A token that can be used to correlate all work performed by the application. All connections associated with the application are established using this correlation token. The value is stored in the DB2 special register CURRENT CLIENT_CORR_TOKEN

These special register can be inquired using SQL, as shown in this example:

```
SELECT
SUBSTR(CURRENT CLIENT_ACCTNG,1,15),
SUBSTR(CURRENT CLIENT_APPLNAME,1,15),
SUBSTR(CURRENT CLIENT_USERID,1,15),
SUBSTR(CURRENT CLIENT_WRKSTNNAME,1,15)
FROM SYSIBM.SYSDUMMY1;
```

But these info fields cannot be set using SQL. They can be changed using connection properties, language APIs, or by calling DB2 provided stored procedure WLM_SET_CLIENT_INFO. The following DB2 for z/OS client special registers can be changed:

- CURRENT CLIENT_ACCTNG
- CURRENT CLIENT_USERID
- CURRENT CLIENT_WRKSTNNAME
- CURRENT CLIENT_APPLNAME

Much of this information is externalized in various forms: the DSNV437I message of the DISPLAY THREAD command, THREAD-INFO data in various messages such as DSNT375I, and the QWHC trace record correlation header, for example.

The improvements introduced by DB2 11 in this area are:

- Expansion of the length of some Client information fields
- Introduction of a new Client information field: CLIENT_CORR_TOKEN

This table shows the changes introduced in DB2 11:

Field name	Max length in DB2 10	Max length in DB2 11
Client User ID	16 bytes	128 bytes
Client Application Name	32 bytes	255 bytes
Client Accounting Information	200 bytes	255 bytes
Client Workstation Name	18 bytes	255 bytes
Client Correlation Token	N/A	255 bytes

The longer client information length provide better granularity in the exploitation of this information, and compatibility with other databases member of the DB2 family of products. The longer client info strings are exploited in:

- WLM enclave classification
- RRSAF DSNRLI SET_CLIENT_ID function
- Rollup accounting
- Profile Monitoring for remote threads and connections
- Resource Limit Facility (RLF)
- Client information special registers
- DISPLAY THREAD command output
- Various trace records, like IFCID 172, 196, 313, and 316
- Various messages that present thread-info

The stored procedure WLM_SET_CLIENT_INFO is enhanced in DB2 11 to support the longer field lengths.

Once DB2 is in new function mode both the resource limit tables and connection profiles can exploit longer values.

The resource limit tables can be used to limit the amount of resources used by SQL statements that run on middle-ware servers. Statements can be limited based on client information, as Application name, User ID, Workstation ID, and IP address. Resource limits apply only to dynamic SQL statements.

The RLF table DSNRLMTxx columns are changed in DB2 11 to support the longer lengths for client information fields, as summarized in this table.

Column name	DB2 10	DB2 11	Comment
RLFEUAN	CHAR(32)	VARCHAR(255)	Specifies an application name
RLFEUID	CHAR(16)	VARCHAR(128)	Specifies an end user's user ID
RLFEUWN	CHAR(18)	VARCHAR(255)	Specifies an end user's workstation name
RLFIP	CHAR(254)	CHAR(254)	The IP address of the location where the request originated

The DDL use to create the RLMT table (DSNRLMTxx) are provided in the DB2 installation job DSNTIJSG.

Profile tables can be used to monitor connections for remote TCP/IP access to DB2 servers. You can use longer client information values to create more granular thresholds to manage and control the use of system resources by particular clients, applications, and users.

Longer fields can be set in any mode (CM / ENFM / NFM) and actually setting the values is not controlled by APPLCOMPAT. However, to retrieve the longer new values using SQL does require new function mode (NFM) with APPLCOMPAT=V11R1, otherwise only short values are retrieved.

Utilizing longer fields using both RLF and Profile can be exploited once DB2 is in new function mode.

CANCEL THREAD FORCE OPTION

In previous versions of DB2 for z/OS, the DDF cancel command did not work to cancel hung threads. The DB2 for z/OS CANCEL THREAD is a reactive command by design: a DB2 thread is only flagged as being cancelled. The thread processing continues until it reaches a cancel detection point where the thread reacts by abnormally terminating itself. These cancel detection points are numerous and strategically distributed in the DB2 code. This reactive cancel behaviour is usually sufficient and successful, but there are cases where the continued processing of the thread may be such that a cancel detection point may not be encountered in a timely manner, or not at all.

A new z/OS function (in z/OS 1.13 or above) allows to terminate DBAT related SRB and Enclave.

The CANCEL THREAD command in DB2 11 includes a new FORCE option, as shown in this example:

```
>>-CANCEL--+-THREAD(token)----------+--+------+--+-------+----->
           '-DDF THREAD(-+-luwid-+-)-'  '-DUMP-'  '-LOCAL-'
                         '-token-'

>--+-----------+--+-------+----------------------------------><
   '-NOBACKOUT-'  '-FORCE-'
```

Use the FORCE option to instruct DB2 to attempt to purge the thread of a remote connection in the DB2 server. The FORCE option is accepted only after a request to CANCEL THREAD is issued without the FORCE option.

DDF PERFORMANCE IMPROVEMENTS

DB2 11 introduces package-based continuous block fetch. It can improve performance and thread utilization for retrieval of large, read-only result sets being accessed from an application running on z/OS accessing a remote DB2 for z/OS server.

Package-based continuous block fetch causes fewer messages to be transmitted from the requester to retrieve the entire result set.

The new package-based continuous block fetch is more efficient than SQL-based continuous block fetch. With package-based continuous block fetch, the requester opens a secondary connection to the DB2 server for each read-only cursor. The DB2 server returns extra query blocks until the all rows for the cursor have been retrieved. When the cursor is closed, the secondary connection is implicitly closed.

In addition, package-based continuous block fetch is easier to configure than the previously existing SQL-based continuous block fetch: it requires only that you bind your applications with the new DBPROTOCOL(DRDACBF) option. You do not need to modify your applications or set subsystem parameters to indicate the maximum number of blocks to be returned for a remote request. This change is available in NFM and requires APPLCOMPAT = V11R1 to be set.

This example shows how to REBIND a package with the new option DBPROTOCOL(DRDACBF):

```
//SYSTSIN DD *
 DSN SYSTEM(DB2P)
 REBIND PACKAGE(DSNTEST.DSNABC) DBPROTOCOL(DRDACBF)
/*
```

Package-based continuous block fetch provides a performance advantage for a DB2 for z/OS application with the following characteristics:

- The application queries only remote sites
- The application does not contain INSERT, UPDATE, DELETE or MERGE statements
- No statement in the application creates a unit of recovery on the remote site

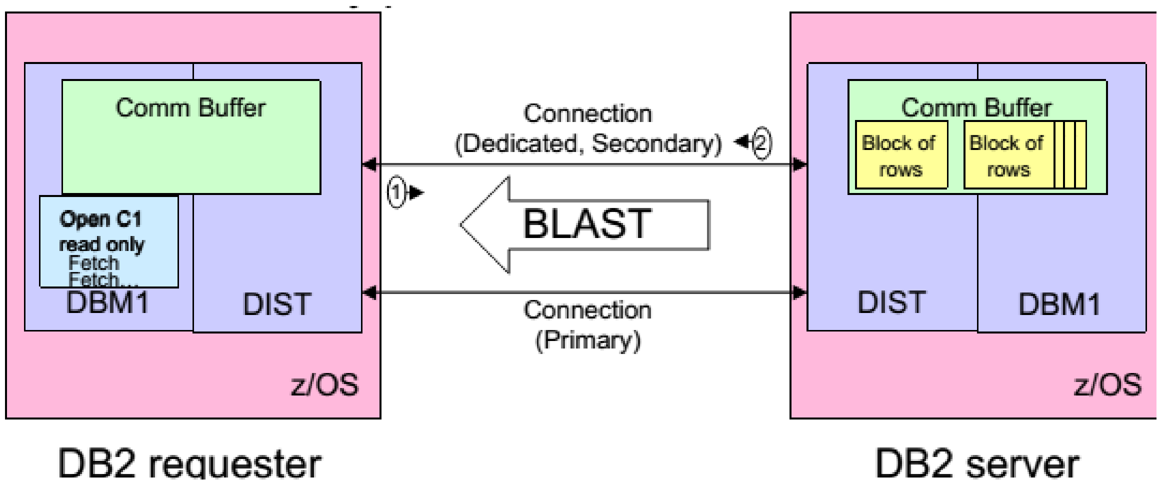


Figure 1 - DBPROTOCOL(DRDACBF)

Some Preliminary internal lab performance results shows the following benefits:

Delta %	Class 1 Elapsed Time	Class 2 Elapsed Time	Class 1 CPU Time	Class 2 CPU Time
Server	-29.5	-8.3	-20.0	-5.8
Requester	-31.1	-31.1	-13	-13

The new exploitation of the synchronous receive model allows DDF to be notified of any socket failure, even while DDF is in sync receive mode. This means that DDF can abort threads running synchronously and have a reduced network latency and CPU in DIST address space. There is also no need to change or even bind the application, it is available from CM and preliminary measurements using the distributed IRWW suite shows both ITR and CPU improvements of 6% to 8% (ITR improvement) and 4% to 6% (CPU improvement).

Please be sure to have the PM80004 APAR for TCPIP to exploit this important feature.

BUFFER POOL ENHANCEMENTS

Bufferpools are a key element in the performance of DB2 subsystems. As systems grow in size, so do bufferpools, and changes have been made for DB2 to handle them more efficiently.

In DB2 version 8.1 bufferpool PGFIX(YES) was introduced to allow permanent fixing of page frames under DB2's control. That was to eliminate the overhead for DB2 fixing and unfixing of each page frame for the brief period it was being used, and removing the possibility of Translation Lookaside Buffer misses requiring z/OS system paging to get the required pages available. Assuming you have enough real storage available to back the bufferpools and you have negligible z/OS system paging, PGFIX(YES) is recommended to be used for bufferpools with a high number of pages per second read into or written out from the bufferpool. Page-fixing of bufferpools has given CPU savings of 5% or more for the DBM1 address space in some subsystems.

Before DB2 10 the page frame sizes for bufferpools were always 4K, but then a 1MB size was available with page-fixing. DB2 11 takes that further by introducing a page-fixed 2GB frame size as well. Hence DB2 subsystems with very large bufferpools on LPARs with very large amounts of real storage (and the latest hardware, microcode and operating system software) can take advantage of the 2GB frame size. Only a small number of sites could benefit from this currently, but in the future there will presumably be many more.

Curiously, DB2 11 also allows 1MB page frame size without page-fixing. This does not appear to offer a significant benefit, hence we don't recommend using that.

DB2 has previously allocated bufferpool buffers as it needs them, building up to the size specified as VPSIZE for each bufferpool. In DB2 11 it does that much faster because it allocates the entire bufferpool space at start-up according to VPSIZE. Also new in DB2 11 you can let WLM manage the size of bufferpools autonomically, based on workload goals and the availability of storage in the system. To do that for any bufferpool you can specify AUTOSIZE(YES), and also specify the minimum and maximum allowed sizes in parameters VPSIZEMIN and VPSIZEMAX or let them default to 75% to 125% of VPSIZE. This takes the setting of the size out of DB2's control, so be cautious about implementing this and monitor what WLM does to your bufferpool sizes.

DATA SHARING ENHANCEMENTS

CASTOUT

With a very high rate of updates, pages could be written to a GBP faster than the castout engines can clear it, possibly resulting in a GBP full condition, which would impact application response times.

DB2 11 has faster castouts by overlapping disk write out I/Os and also by reducing the size of the notify messages sent to castout owners

RESTART LIGHT

After an abend a DB2 group member may be restarted using RESTART LIGHT, so that it starts quickly and releases most of the retained locks, but does no castout processing, hence page set P-locks in IX or SIX mode are not released. This can block utilities and impact DB2 availability.

A DB2 11 group member can be started: START DB2 LIGHT(CASTOUT) to release retained locks but also do castout processing to release the page set P-locks in IX or SIX too. Therefore, utilities are not blocked. This does take a little longer than the basic restart light, but it improves DB2 availability.

LPL PAGES

If a DB2 data sharing group member suffers a disorderly shutdown and is restarted there can be pages put into LPL status. Then all the affected tablespaces have to be identified and a manual START DB() SP() command is required to individually recover each of them, which can delay full availability of the DB2 data for longer than desirable.

DB2 11 automatically recovers all pages which have been put into LPL during a DB2 restart. This is comparatively fast and reliable, improving availability and reducing human effort. This is another worthwhile enhancement.

SELECT FROM DIRECTORY PAGESETS

DB2 for z/OS maintains a set of tables (in database DSNDB01) called the DB2 directory. Historically, those tables were not accessible through SQL.

In V10 originally selects was added for SYSIBM.SYSLGRNX, SYSIBM.SYSUTIL, SYSIBM.SYSUTILX and recently in APAR PM80685 added selects from the following tables to v10: SYSIBM.DBDR, SYSIBM.SPTR

V11, added SYSIBM.SCTR table added to the list.

Let's look closely.

- SYSIBM.DBDR: The DBDR table stores one row for each DBD section.
- SYSIBM.SCTR: The SYSIBM.SCTR table stores Skeleton Cursor Tables (SKCT) information
- SYSIBM.SPTR: The SYSIBM.SPTR table stores Skeleton Package Table (SKPT) information
- SYSIBM.SYSLGRNX: The SYSLGRNX table stores recovery log ranges that record the time an indexspace defined with COPY YES or a table space was open for updates.
- SYSIBM.SYSUTIL: The SYSUTIL table stores status information about DB2 utilities that are active or stopped.

Some of the data in those tables are still internal but combining them to existing catalog tables, provide us more information.

For instance, joining SYSIBM.DBDR Directory table and SYSIBM.SYSDATABASE catalog table provides the name of DATABASESs with the total number of sections. That number gives an idea about the size of DBD.

```
---------+---------+---------+---------+------
    SELECT NAME,COUNT(*) AS NUMBER_OF_SECTIONS
    FROM SYSIBM.DBDR A, SYSIBM.SYSDATABASE B
    WHERE A.DBID = B.DBID
    GROUP BY NAME
    ORDER BY NUMBER_OF_SECTIONS DESC;
---------+---------+---------+---------+------
NAME                      NUMBER_OF_SECTIONS
---------+---------+---------+---------+------
DSNDB06                                   12
DGOLD107                                   8
DANLDBU                                    4
SEMTPDB1                                   2
DSNOPTDB                                   2
DSNRGFDB                                   1
MGBTEST                                    1
MGBMAP                                     1
```

Take into account that the actual data and contents, while being able to be queried by SQL, are in internal format and in BLOB so they are not currently formattable.

For example:

```
SELECT SECLEN, DBID, DBD_DATA FROM SYSIBM.DBDR ;
-------+---------+---------+---------+---------+---------+---------+--
   SECLEN    DBID   DBD_DATA
-------+---------+---------+---------+---------+---------+---------+---
     8000       1   20391F40C4C2C44000000000C4E2D5C4C2F0F140000100FF00
```

ADDITIONAL ZIIP ENABLEMENT

zIIP usage in DB2 has grown up since its' first days with DB2 V8. There were many upgrades to its capabilities during the years and with each new DB2 version zIIP usage has gone up. DB2 11 is no exception – more work is done on zIIP than ever before. The demand for zIIP usage has never been that high as nowadays.

DB2 FOR Z/OS AND SPECIALTY ENGINES

An IBM System z server can be configured with optional processors known as Specialty Engines. Specialty engines are slightly modified standard processors that are designed to offload eligible workloads from General Purpose processors. Some of the CPU processing that would otherwise be executed on general processors can, under certain conditions, be executed on a specialty engine.

At the moment of this writing, four specialty engines are currently available:

- System z Integrated Information Processor (zIIP)
- System z Application Assist Processor (zAAP)
- Integrated Facility for Linux (IFL)
- System Assist Processor (SAP)

z/OS Integrated Information Processors (zIIPs) can help to reduce dramatically Total Cost of Ownership for System z mainframes because the workload executed on them is not taken into account for software pricing. In addition, the purchase price of a zIIP is typically significantly lower than the price of a standard processor.

zIIPs were introduced in 2006 as part of System z9 hardware and designed to work with DB2 for z/OS version 8. Since then, the number of DB2-eligible workloads has been growing steadily. Several zIIP-related improvements were provided with DB2 10 and DB2 11 for z/OS. DB2 for z/OS is not the only software that can leverage the ability to offload CPU to zIIPs; many software products from IBM and other vendors can take advantage of these specialty engines, as well.

Specialty engines are a highly effective way of reducing CPU costs by offloading a part of CPU execution from a General Purpose processor to a special processor. DB2 11 continues the trend of increasing the zIIP value by expanding the DB2 CPU zIIP eligibility for utility and system tasks workloads.

A zIIP processor is not required to evaluate its potential benefits. The PROJECTCPU=YES PARMLIB parameter enables z/OS to collect Specialty Engine usage as if a Specialty Engine was installed. This projection capability can be run at any time, including on a production environment. PROJECTCPU can help to determine the number of zAAP and zIIP engines required to satisfy a specific customer's workload.

The z Application Assist Processor (zAAP) specialty engines are similar to zIIPs, but they are dedicated to running specific Java and XML workloads on z/OS. Version 1.11 of z/OS introduced a feature that allows users to run zIIP and zAAP eligible workload on installed zIIP processors. This feature is enabled through the use of the ZAAPZIIP keyword in the IEASYSxx member of SYS1.PARMLIB. When ZAAPZIIP is set to YES, zAAP eligible work is able to run on an available zIIP processor when no zAAP processors are present.

Please note that IBM is deprecating the zAAP engine and incorporates it into the zIIP engine. As of the announcement on zBC12 the ratio of 1 zIIP & 1 zAAP per 1 CP is changed to 2 zIIP engine as part of that change.

DB2 11 AND ZIIPS

DB2 11 directs more tasks to be run under enclave SRBs, which has increased the amount of DB2 MSTR and DBM1 processing that IBM authorizes to execute on a zIIP specialty engine. The following is DB2 11 for z/OS processing that will be authorized to execute on a zIIP:

- Asynchronous processing that is executed under enclave SRBs (Service Request Blocks) and that will be "charged" for CPU consumption purposes to an IBM DB2 address space (rather than to user applications), with the exception of P-lock negotiation processing. Such zIIP eligible processing includes:
    - Cleanup of pseudo deleted index entries as part of DB2 system task cleanup[1]
    - Cleanup of XML multi-version documents (available in DB2 10 for z/OS via APAR PM 72526)
    - Log write and log read
- DB2 base LOAD, REORG and REBUILD INDEX utility processing of inline statistics collection (up to 30% offload)
- DB2 base processing of RUNSTATS utility Column Group Distribution statistics collection (up to 80% offload)
- DB2 base LOAD utility index management processing when running LOAD REPLACE

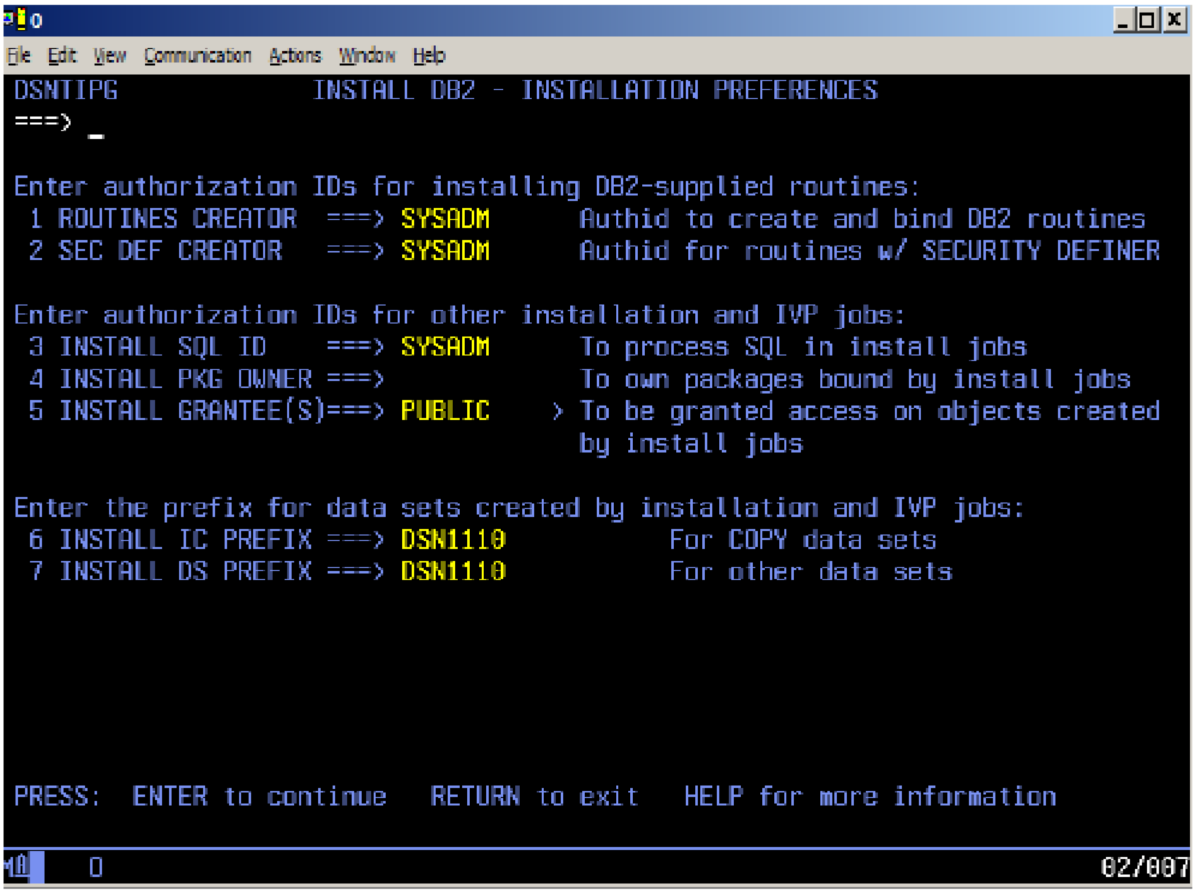
[1] Governed by the combination of the new INDEX_CLEANUP_THREADS ZPARM (which contains the number of parallel cleanup threads) and the contents of SYSIBM.SYSINDEXCLEANUP table (which dictates which indexes are governed and when to do the cleanup).

INSTALLATION

If you have come to know and love the original DSNTINST CLIST, you will be pleased to know that it continues to be the way to generate the batch jobs you will need for DB2 installation or version migration, though there are some enhancements to it for DB2 11.

Previously it generated many batch jobs in a new SDSNSAMP library, then the human installer would often need to manually modify some parts of them before running, to comply with their local installation standards. That meant that rerunning DSNTINST could easily lose those changes by regenerating the job members.

To reduce that manual updating DB2 11 install includes new panel DSNTIPG (see Figure 2 below) to include many of those common customisations. Particularly affected were the authority ids used for dynamic SQL, and most of those are now customisable via a new panel DSNTIPG, and those IDs are implemented in the jobs via OWNER(authid) parameters in BINDs or SET CURRENT SQLID statements.



```
DSNTIPG              INSTALL DB2 - INSTALLATION PREFERENCES
===> _

Enter authorization IDs for installing DB2-supplied routines:
 1 ROUTINES CREATOR   ===> SYSADM      Authid to create and bind DB2 routines
 2 SEC DEF CREATOR    ===> SYSADM      Authid for routines w/ SECURITY DEFINER

Enter authorization IDs for other installation and IVP jobs:
 3 INSTALL SQL ID     ===> SYSADM      To process SQL in install jobs
 4 INSTALL PKG OWNER ===>              To own packages bound by install jobs
 5 INSTALL GRANTEE(S)===> PUBLIC     > To be granted access on objects created
                                       by install jobs

Enter the prefix for data sets created by installation and IVP jobs:
 6 INSTALL IC PREFIX ===> DSN1110            For COPY data sets
 7 INSTALL DS PREFIX ===> DSN1110            For other data sets




PRESS:  ENTER to continue   RETURN to exit   HELP for more information

MA      0                                                           02/007
```

Figure 2 - New Install Panel

That panel also newly allows for different dataset name prefixes for some of the generated libraries. That is good, but it still does not let you generate install jobs for DB2 subsystems which will use input libraries which either do not yet exist or they exist on a different z/OS system (where the install jobs will be run); you are forced to supply dataset names that exist on the current system, then manually edit all the generated jobs to match the actual dataset names.

DSNTINST still saves most of your installation parameter values only after you finish entering values on the last panel, when it generates a fresh DSNTIDxx parameter member in your SDSNSAMP library.

User tip #1: DSNTINST error handling is not robust, hence it can crash, losing the values you have entered; therefore it is recommended to use the SAVE command (which was new with DB2 10) occasionally in the dialog to preserve what you have entered – storing most of your parameter values into your DSNTIDxx output member.

User tip #2: DSNTINST stores some parameter values in the current ISPF profile pool. Therefore, if you are installing multiple DB2 subsystems in the same parallel sysplex – invoke the dialog using a different ISPF application id for each DB2 subsystem like:

```
SELECT CMD( EXE 'prefix.SDSNCLST(DSNTINST)' …. ) NEWAPPL(ssid)
```

The output DSNTIDxx member contains many parameters used to generate the DSNTIJUZ member in your NEW.SDSNSAMP library, and the DSNTIJUZ job creates the DSNZPARM and DSNHDECP modules in your SDSNEXIT library. You can run DSNTINST just to create a fresh DSNTIJUZ job if you want. Many parameters in the DSNTIDxx member were different from the equivalent macro parms in the DSNTIJUZ job, and the good news is that the new (ZPARM) parameters are called the same in both places now, though the old ZPARM parameters are still mostly different in the 2 members. Unfortunately, not all valid parameters are known in DSNTINST, hence in many cases after it is first generated for DB2 11 you will still need to manually update and maintain your DSNTIJUZ job rather than just generating it again.

MIGRATION

The migration process to DB2 11 is very similar to that migrating to 10, except there is no skip migration allowed – you can migrate to DB2 11 only from DB2 10 NFM.

One prerequisite for migrations has been modifying existing applications to comply with standards for the new version before migrating. That could mean modifying the SQL for new reserved words or for changes in SQL functions.

DB2 11 overcomes that problem by allowing separation of system and application migration, via the new **APPLCOMPAT** ZPARM to make BIND/REBIND behave as in DB2 10. That also permits a fallback of the subsystem from DB2 11 CM mode to DB2 10 NFM mode, without needing to REBIND any packages bound in DB2 11.

However, before migration to DB2 11 you must rebind ALL packages on DB2 10 which were last bound in version 9 or before.

Additionally, there were a few ZPARMS in DB2 10 which allowed continued use of DB2 behaviour from version 9 (or earlier) and they continue to work in DB2 11.

- **RRF** can be set to "DISABLE" to keep the creating new tables rows in Basic Row Format rather than Reordered Row Format.
- **BIF_COMPATABILITY** can be set to "V9" to keep the old (non ANSI compliant) formatting of decimal data by CHAR or VARCHAR built-in functions.

Never forget that DB2 will not support old standards indefinitely, and some changes are required to get the benefits of performance enhancements, hence the general advice is to upgrade your systems to comply with new standards as soon as convenient.

Sometimes you may not be sure of the ZPARM values of a subsystem to be migrated. Since version 9 IBM has recommended using DSNTIJXZ job to generate a SDSNSAMP(DSNTIDxx) parameter member with the current ZPARMs which you can then use as input to the DSNTINST migration CLIST. However, the generated member still must be treated as an aid rather than 100% reliable – some parameters are set to default values and some (e.g. some WLM parameters) can be incorrect. That has not been resolved for migrating to version 11.

EXPLOITATION OF EC12 FEATURES

IBM announced the latest generation zEC12 enterprise servers in August 2012, with up to 101 configurable processors per server, each running at an industry-leading 5.5GHz. In addition to an impressive list of general performance and capacity improvements over the previous-generation z196 enterprise servers, the zEC12 models include a number of features which will be specifically exploited by DB2 11.

2GB PAGE FRAMES

DB2 10 for z/OS introduced support for 1MB "large page frames", an enhancement designed to reduce processing overheads for very large DB2 buffer pools by allowing z/OS to manage the underlying real storage in fewer 1MB pieces rather than many more 4KB pieces.

1MB page frames required the associated buffer pools to be fixed in storage (via the PGFIX=YES buffer pool attribute). They also required an area of real storage to be reserved specifically for large pages at IPL time (set by the LFAREA parameter of IEASYSnn in PARMLIB).

Many customers with larger DB2 buffer pools were able to achieve CPU savings of up to 4% by exploiting this capability. However, as memory prices fall and workloads continue increase, DB2 buffer pools also increase in size and the z/OS overheads of managing even the larger 1MB page frames start to become more significant.

In recognition of these trends, when running on an zEC12 server DB2 11 will support even larger 2GB page frames, each of which will map onto more than half a million 4K pages as shown in the bottom example in the diagram on the right. New options on the z/OS LFAREA parameter allow you to specify the amount of real storage to be reserved for both 1MB and 2GB large page frames (also needs z/OS APAR OA40967). Note that buffer pools backed by 2GB page frames must be page-fixed, as with the 1MB page frame support in DB2 10.

Those customers using very large DB2 buffer pools will see further CPU reductions by moving to 2GB page frames, due to reductions in the z/OS Translation Look-aside Buffer (TLB) overheads associated with managing fewer pages. Other sites may not have sufficiently large pools for 1MB page frames to be a significant limitation today, but that situation will undoubtedly change in the future as buffer pool sizes continue to grow. By moving early to support 2GB page frames IBM has recognised and eliminated an important future scalability issue.

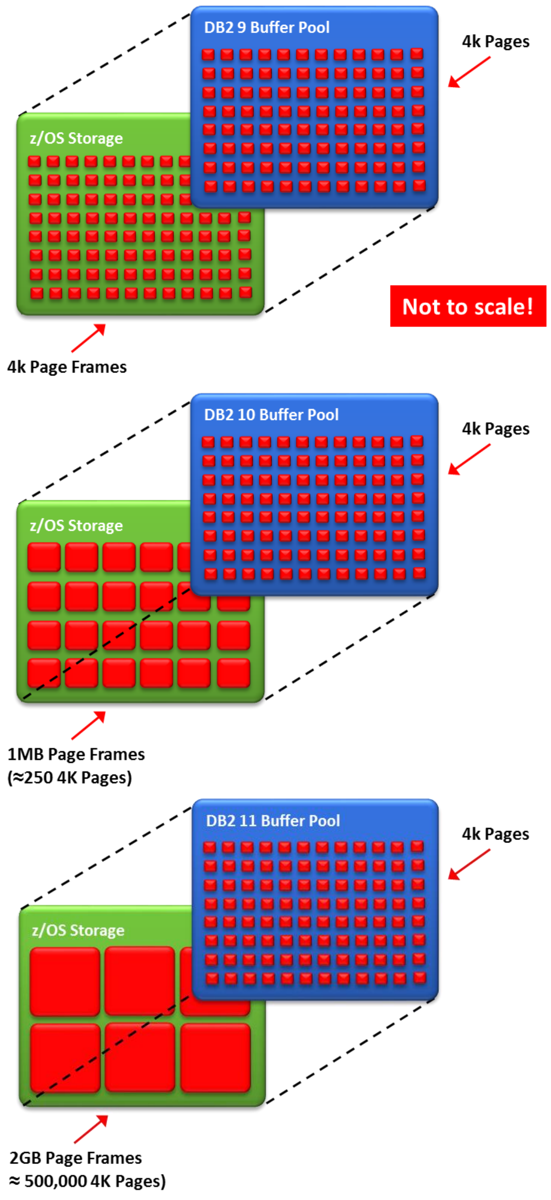


Figure 3 - Large Page Frame Support

PAGEABLE 1MB PAGE FRAMES

As stated above, the 1MB page frame support introduced in DB2 10 for z/OS required that the associated buffer pool was page-fixed. Although page-fixing buffer pools with a high I/O intensity can pay some significant CPU dividends, this pre-requisite made it difficult for some customers to make use of 1MB page frame support (either due to legitimate concerns over the impact of page-fixing a significant proportion of the available real storage on an LPAR or, just as commonly, due to z/OS systems programmers objecting to page fixing as a matter of principle!).

For those customers running DB2 11 on a zEC12 server with the Flash Express feature[2], it is now possible for non-page-fixed buffer pools to be partially backed by 1MB page frames (this feature is also being retro-fitted to DB2 10 via APAR PM85944). Although you'll obviously miss out on the CPU benefits associated with page-fixing a buffer pool with a high I/O intensity, this new option could still provide some performance benefits due to reductions in TLB overheads.

DB2 CODE USING LARGE PAGE FRAMES

As discussed in the previous section DB2 10 and DB2 11 have exploited 1MB and 2GB large page frames to allow more efficient handling of large buffer pools. However, despite the extensive use of large memory objects in the past few releases of DB2, the storage used for DB2 code (as opposed to the data held in buffer pools) remained backed by standard 4K page frames.

DB2 11 is able to utilise large page frames for DB2 code objects and log output buffers, in addition to buffer pools (requires z/OS 2.1). This enhancement can significantly reduce the z/OS overheads associated with DB2 logging and code management, reducing CPU consumption and lowering operational costs.

SUMMARY

With the ability to support 4K, 1MB and 2GB page frames, there are now lots of options for configuring DB2 real storage and each has its own set of benefits and pre-requisites. The table below summarises these.

Frame Size	Page Fix	Supported By	Hardware Requirement	Benefit
4K	NO	All	N/A	Most flexibility
4K	YES	All	N/A	CPU reduction during I/O
1M	NO	DB2 10 with PM85944 DB2 11	zEC12 and Flash Express	CPU reduction from TLB hit
1M	YES	DB2 10 and above	z10 and above LFAREA 1M=xx	CPU reduction during I/O, CPU reduction from TLB hit
2G	YES	DB2 11	zEC12 LFAREA 2G=xx	CPU reduction during I/O, CPU reduction from TLB hit

SEE ALSO

Please see below for a cross reference to other topics that may also be of interest to systems programmers, but have been placed elsewhere within this document.

[2] Flash Express offers up to 6.4TB of CPU-attached SSD (Solid State Disk), essentially forming an intermediate tier between real storage and external disk subsystems.

- BIND/REBIND Enhancements on page 29
- SQL Compatibility Feature on page 32
- Security Enhancements on page 48
- Instrumentation Enhancements on page 53

DBA TOPICS

This section of the document discusses new DB2 11 features that are expected to be of most interest to DB2 Database Administrators.

TEMPORAL ENHANCEMENTS

DB2 customers have been managing temporal data stores for many years utilizing home-grown designs and techniques. This often involved creating complex table designs, history tables, and writing application logic and/or database triggers to support temporal controls and time travel queries. This all changed significantly with DB2 10 for z/OS and the introduction of automated temporal tables and time travel queries. DB2 10 introduced two types of temporal tables: application-period and system-period. The application-period temporal tables are useful for the management of data that is active during a certain time period, such has insurance policy information, and allows data to be staged (active in the future), current, and historical in nature. The system-period temporal tables are useful for tracking physical changes to data values, which makes them great for auditing and/or transaction history, and thus important for compliance and quality control and analytics. Certain database automation was put in place in order to simplify the creation and maintenance of temporal tables. Application developers can incorporate a time dimension in their design without having to code for the maintenance of the time factor, and can rely on DB2 to maintain the proper consistency of their time dependent data. For system-period temporal tables DB2 could optionally maintain data in a history table in order to track all changes to the data over time.

NEW SPECIAL REGISTERS TO SUPPORT TIME TRAVEL QUERIES

One of the challenges that was evident early on as people began experimenting with DB2 time travel queries was that not all of the application programming was eliminated. That is, while DB2 provided for a period-specification in a FROM clause values still needed to be provided by the application developer coding the SQL statement. This proved to be a bit of a challenge for application developers as they quite often had to code separate SQL statements for non-period-specific and period-specific queries. Any easy way to visualize this is if a program needed to access data from the EMP system period temporal table that had a definition such as the following:

```
CREATE TABLE T_EMP (
EMPNO   CHAR(6) FOR SBCS DATA NOT NULL,
START_TS       TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
END_TS         TIMESTAMP(12) GENERATED ALWAYS AS ROW END   NOT NULL,
TRANS_TS       TIMESTAMP(12) GENERATED ALWAYS
AS TRANSACTION START ID IMPLICITLY HIDDEN,
LASTNAME       VARCHAR(15) FOR SBCS DATA NOT NULL,
WORKDEPT       CHAR(3) FOR SBCS DATA WITH DEFAULT NULL,
PERIOD SYSTEM_TIME(START_TS, END_TS));

CREATE TABLE T_EMP_HIST LIKE T_EMP;
ALTER TABLE T_EMP
ADD VERSIONING USE HISTORY TABLE T_EMP_HIST;
```

In order to access the “current” or “as of now” data in the T_EMP table for employee “000010” the developer would code a query that looked like this:

```
SELECT LASTNAME
FROM   T_EMP
WHERE  EMPNO = '000010'
```

Then, if the data for employee “000010” was desired “as of” a specific point in time then a second query would be coded:

```
SELECT LASTNAME
FROM   T_EMP FOR SYSTEM_TIME AS OF ?
WHERE EMPNO = '000010'
```

The parameter marker in the above example could also have been a value, expression, or host variable. So, application developers found themselves coding different sets of queries for both types of temporal tables (application-period and system-period) in order to find the appropriate data depending upon the business needs. This turned some people away from implementing a DB2 based temporal design. Please keep in mind that even if the value in the temporal query above is set to CURRENT TIMESTAMP it is still a temporal time travel query. After all “right now” is “just then” right after “right now” has passed! So CURRENT TIMESTAMP is a point in time that cannot be considered “current”, “active”, or “as of now” for a period-specification. This applies to both application-period and system-period temporal tables, and thus also to bi-temporal tables.

DB2 11 for z/OS improves on the concept of time travel queries by automating the inclusion or exclusion of period-specifications in a FROM clause by means of two new special registers.

- CURRENT TEMPORAL BUSINESS_TIME
- CURRENT TEMPORAL SYSTEM_TIME

By default the value of these special registers is the null value. If the package bind option SYSTIMESENSITIVE for system-period or BUSTIMESENSITIVE for application-period is set to YES, which it is by default, then these special register values affect the activation of the period-specification in FROM clauses of temporal tables automatically. Using these special registers allows for only one query to be coded against a temporal table. Whether or not that query is a normal query or time travel query is then dependent upon the special register setting. Setting the special register to the null value will cause the query to execute as if there was no period-specification, and any valid non-null value will cause the query to execute as an “as of” time travel query. For example instead of coding two separate queries for our T_EMP system-period temporal employee table we only code one statement:

```
SELECT LASTNAME
FROM   T_EMP
WHERE  EMPNO = '000010'
```

If the CURRENT TEMPORAL SYSTEM_TIME special register is set to the null value, which it is by default, then the above statement executes as is. If the CURRENT TEMPORAL SYSTEM_TIME special register is set to any other valid value then the above statement executes as if it was coded as follows:

```
SELECT LASTNAME
FROM   T_EMP FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME
WHERE  EMPNO = '000010'
```

This obviously eliminates having to code multiple statements in an application to handle the need for time travel queries and greatly simplifies application development of applications that access temporal tables. Please keep in mind that only a null value for the special register inactivates the time travel aspect of these queries. There are also potentially dramatic performance improvements for temporal queries as a result of using this new coding technique. This performance potential is documented in this white paper in the section entitled *Extended Access Paths for "As of Now" Time Travel queries*. See that section for results of some of the tests conducted using the CURRENT TEMPORAL SYSTEM_TIME special register.

EXTENDED ACCESS PATHS FOR "AS OF NOW" TIME TRAVEL QUERIES

System-period temporal tables are a powerful feature that was introduced with DB2 10 for z/OS. Having the DB2 automation replace significant quantities of application programming was a huge advantage with regards to simplifying the application development process, and reducing time to delivery for a solution. Customers began implementing system-period temporal base tables and history tables, and coding time travel queries against these tables. One thing that became evident quite quickly, however, was that always using a time travel query to retrieve data could become a challenge from a performance perspective. This was because a system-period time travel query is always transformed into a UNION ALL between the base table and the history table. Therefore, even if "as of now" data was desired both the base and history table were accessed. This performance impact became even more significant with table joins, as joins have to be distributed across base and history table. So, what appeared to be a two table join between two system-period temporal tables with period-specifications actually became a join of two tables four times for a total access of eight tables. Given that current data represented the majority of typical access a performance hit was taken 100% of the time. Application developers had to code separate queries to retrieve current data versus historical data in order to maintain an acceptable level of performance.

Imagine a system-period temporal table call T_EMP. If a period-specification is used to access the table such as the following query:

```
SELECT LASTNAME
FROM   T_EMP FOR SYSTEM_TIME AS OF CURRENT TIMESTAMP
WHERE  EMPNO = '000010'
```

Then the query, after the query transformation component of DB2 has transformed it, would look like this:

```
 (SELECT LASTNAME FROM T_EMP WHERE ( T_EMP.EMPNO = '000010'
 AND T_EMP.START_TS <= CURRENT TIMESTAMP
 AND T_EMP.END_TS > CURRENT TIMESTAMP)
UNION ALL
 SELECT LASTNAME FROM T_EMP_HIST WHERE ( T_EMP_HIST.EMPNO = '000010'
 AND T_EMP_HIST.START_TS <= CURRENT TIMESTAMP
 AND T_EMP_HIST.END_TS > CURRENT TIMESTAMP))
```

If all the application wanted was current data then if would have to access the T_EMP table directly without a period-specification. That is the only way the UNION ALL could be avoided.

With the introduction of the CURRENT TEMPORAL SYSTEM_TIME special register in DB2 11 for z/OS this potential performance issue is addressed. During statement compile time (prepare for dynamic or bind for static) an extended section is added to the statement access path. So, for queries that do not specify a period-specification DB2 will build two sections for the table access; one without the UNION ALL and temporal predicates, and a second with the UNION ALL and temporal predicates. This will only happen if the package bind parameter SYSTIMESENSITIVE is set to YES, which is the default. The creation of the alternate access paths for dynamic is also dependent upon the package bind parameter for the dynamic package the application is using. Which access path is used at execution time is then dependent upon the setting of the CURRENT TEMPORAL SYSTEM_TIME special register. If the value is set to the null value, then the original section is executed and no UNION ALL or time travel predicates apply. If the value of the special register is set to something other than the null value then the extended section is used, including the UNION ALL and time travel predicates. Since the null value is the default for this special register then an existing table can be converted to a system-period temporal table without application impact at all. Then a small application change to include the setting (or not setting) of the special register can then dictate whether or not the query is a base query or time travel query, again without changes to the original statement. For example, the following statement against a system-period temporal table:

```
SELECT LASTNAME
FROM   T_EMP
WHERE  EMPNO = '000010'
```

Would actually have two separate access paths if SYSTIMESENSITIVE was YES. The first access path would represent the statement as coded, and the second would represent the time travel query:

```
 (SELECT LASTNAME FROM T_EMP WHERE ( T_EMP.EMPNO = '000010'
 AND T_EMP.START_TS <= CURRENT TEMPORAL SYSTEM_TIME
 AND T_EMP.END_TS > CURRENT TEMPORAL SYSTEM_TIME)
UNION ALL
 SELECT LASTNAME FROM T_EMP_HIST WHERE ( T_EMP_HIST.EMPNO = '000010'
 AND T_EMP_HIST.START_TS <= CURRENT TEMPORAL SYSTEM_TIME
 AND T_EMP_HIST.END_TS > CURRENT TEMPORAL SYSTEM_TIME))
```

The value of the CURRENT TEMPORAL SYSTEM_TIME special register would control which access path is used, be it original or extended, and any query that requests "as of now" data using a CURRENT TEMPORAL SYSTEM_TIME set to the null value would not incur the overhead of the UNION ALL. This same performance feature is also available for the new archive enabled tables.

ONLINE SCHEMA CHANGE ENHANCEMENTS

As applications evolve, the associated changes to DB2 objects such as tables and indexes remain one of the most common reasons for disruptive planned outages. Dynamic schema change allows DBAs to alter the definition of DB2 objects while maintaining access to the underlying data, with the change being fully materialised the next time the data is reorganised as part of routine housekeeping.

IBM began focusing on dynamic schema change back in DB2 Version 8, and has steadily expanded its capabilities in every release since. DB2 11 introduces a number of additional enhancements.

DROP COLUMN

Modifying Column Structures has a long history in DB2. Adding a column is implemented in the very early versions. Then Altering Column data type and renaming column name came up with V10. In V11 we have Drop Column functionality.

WHY DO YOU NEED TO DROP COLUMN?

Columns become obsolete as applications change. Could the column be left in the table?

Leaving a column has some cost. For instance real space in every row stored in the table and in every Image Copy of the tablespace. Potential space taken up in the log records written for the table. Additional CPU and elapsed time costs are incurred with in all aspects of accessing and maintaining the data.

Besides all of that, DBA must spend time to remember that the column is redundant/obsolete and Developer must analyze the usage of the column.

HOW DO YOU DROP A COLUMN BEFORE DB2 11

You have to DROP and recreate the Table to remove the obsolete column as shown below. Below is a very preliminary procedure to do that task, which is very sensitive and open to human errors.

- Schedule an outage
- Unload Data
- Drop Table
- Alter DDL
- Create Table
- Load Data
- Grant Authorities and Bind/Rebind Packages

IMPROVEMENTS IN DB2 11

You can only drop a column under certain circumstances. A column cannot be dropped if

- The containing table space is NOT a Universal Table Space
- The table is a created global temporary table, a system-period temporal table, a history table, MQT or table referenced by MQT plus others....
- There are row permissions or column masks dependent on the table
- There are triggers defined on the table
- The column is an XML column, is part of the table partitioning key, RI, plus others...
- Hash Key column
- DOCID / ROWID column
- Tables with EDITPROC or VALIDPROC

In addition, no immediate alters (e.g. change data type, add column, add/rotate parts) can be combined with a pending alter.

HOW DOES IT WORK?

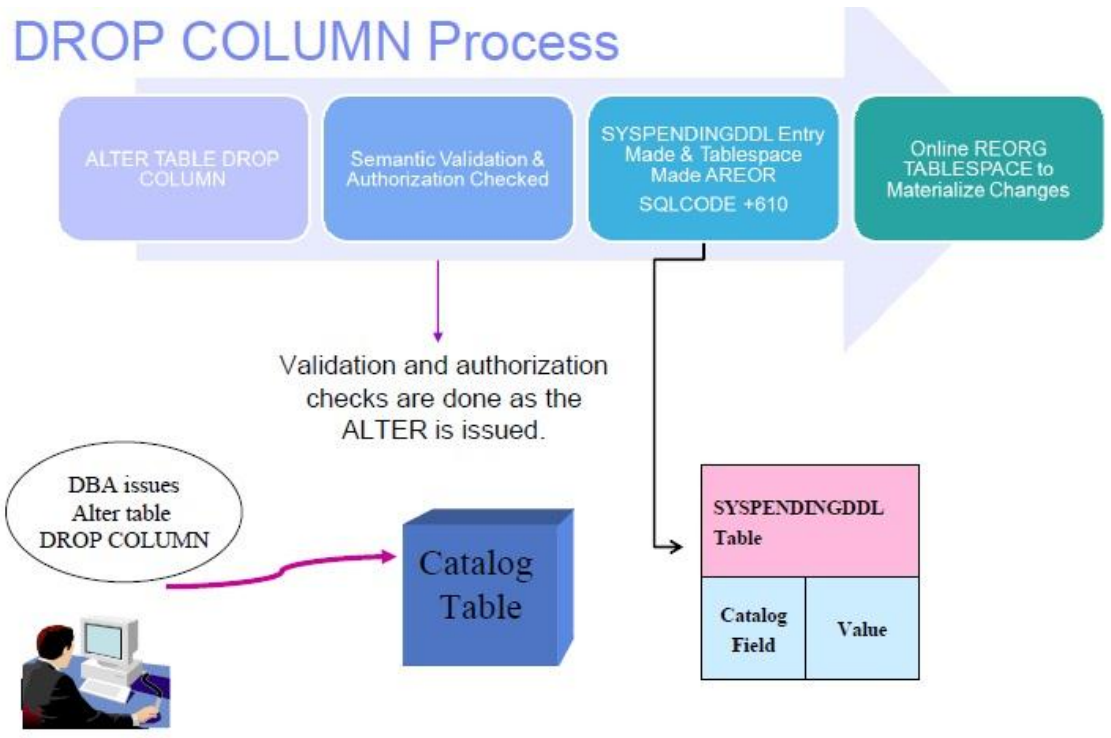

DROP COLUMN Process
ALTER TABLE DROP COLUMN
Semantic Validation & Authorization Checked
SYSPENDINGDDL Entry Made & Tablespace Made AREOR
SQLCODE +610
Online REORG TABLESPACE to Materialize Changes
Validation and authorization checks are done as the ALTER is issued.
DBA issues Alter table DROP COLUMN
Catalog Table
SYSPENDINGDDL Table
Catalog Field
Value

Figure 4 – DROP COLUMN Process

```
ALTER TABLE mytable DROP COLUMN mycolumn RESTRICT
```

Validation and authorization checks are done as the ALTER is issued. SYSPENDINGDDL entry made & tablespace made AREO with SQLCODE +610

```
REORG TABLESPACE Db.ts SHRLEVEL REFERENCE
```

Tablespace with new defined schema.

There are also new SQL Codes coming with the DROP COLUMN Enhancement.

<SQLCODE -195: LAST COLUMN OF table-name CANNOT BE DROPPED> means you must have at least one column after you drop the column you want.

<SQLCODE -196: COLUMN table-name.column-name CANNOT BE DROPPED. REASON = reason-code> this is the most common error you're likely to see. The reasons will be one of the list items in restrictions.

IMPACT ON OTHER UTILITIES

RECOVER

- RECOVER to a PIT prior to a materializing REORG is allowed, but only the data will be recovered to that point in time: the schema definition will remain as it was following the materializing REORG. Please see Recovery support for deferred schema changes on page 27 for more details.
- An inline copy will be taken by the REORG. This single image copy is the only base for recovery

UNLOAD

- will not be able to unload from an Image Copy which contains data for dropped columns
- Dropped columns is no longer exist in the catalog after REORG materialized data
- Consider UNLOAD with external formatting If data corruption occurs, can be used with LOAD utility if necessary

RUNSTATS

- Should be run after materializing REORG. Ensures statistics are current for BIND/REBIND or PREPARE

REORG

- Only table space level REORG materializes the changes

SUMMARY

A long requested DROP COLUMN feature is ready in V11. It has some restrictions but it may save you a lot of outage time when compared with traditional method.

ALTER PARTITION LIMIT KEYS

In Version 11, you can alter the limit keys for a partitioned table space without impacting the availability of the data. This capability is called an online alter limit key.

In previous versions of DB2, when you changed the limit key values, all affected partitions were placed in REORG-pending (REORP) status. The data was unavailable until you ran the REORG utility.

```
ALTER TABLE … ALTER PARTITION integer ENDING AT
```

BEHAVIOUR PRIOR TO DB2 11

- Affected partitions are set to REORP.
- These partitions could not be accessed until reorg.
- Any REORG could be run to redistribute the data and remove the status.

In Version 11, that restriction is removed. When you change the limit key values, the data remains available and applications can continue to access the data. However, the limit key changes are not materialized until the next time that you run REORG. The affected partitions are placed in advisory REORG-pending (AREOR) status.

This type of definition change, where the changes are not immediately materialized, is called a pending definition change.

This new online alter limit key capability is limited to Range-partitioned universal table spaces and table spaces that are exclusively partitioned with table-controlled partitioning.

In addition to improved data availability, online alter limit key provides other benefits. Because altering a limit key is now a pending definition change, you can alter a limit key even when the table space has other pending definition changes.

Also, because the materialization of the limit key changes is deferred, you have a time period where you can drop the limit key changes without affecting the target table (please see Drop Pending Changes on page 29).

ALTER LIMIT KEY IN DB2 11

- Alter limit key is treated as a pending alter in NFM.
- The affected partitions are set to AREOR.
- Online REORG must be run to materialize the pending changes.
- REORG SHRLEVEL NONE does not materialize these changes.
- UTS – partitioned by range (PBR) or Classic partitioned table spaces (table controlled partitioning) is a prerequisite for this feature.
- Objects, in which the partitioning is index controlled, must be first converted!
- The new limit keys are materialized in SYSTABLEPART in the SWITCH phase (new message DSNU2916I)
- If the table space is in a materialized query table relation, it is still possible to alter limit key online.
- RECOVER PIT is allowed, but requires a subsequent REORG due to setting of REORP after the recovery. This is possible but need attention because it is restrictive.

When compared with V10 behaviour:

- REORG SHRLEVEL NONE does not materialize the pending changes.
- LOAD REPLACE does not materialize the pending changes.
- If special REORG keywords are not allowed, message DSNU2917I is issued

SUMMARY

Altering partition limit keys in V11 is an ongoing enhancement since V10. Although it was a long-awaited feature, it was too restrictive to apply for daily life. In V11, applying this feature with enhancements makes it more practical for daily usage.

RECOVERY SUPPORT FOR DEFERRED SCHEMA CHANGES

The deferred schema change capability first introduced in DB2 10 (and further enhanced in DB2 11 as discussed above) allows DBAs to make schema changes at any time but defer the materialisation of those changes until a REORG can be scheduled at a convenient time.

However, the DB2 10 implementation included a significant restriction relating to point-in-time (PIT) recoveries. As shown in the diagram below, once the REORG had been run to materialise the pending change (at T2 in the example) it was not possible to perform a recovery to a prior point in time. DB2 11 NFM removes this restriction, allowing recovery to any valid prior point.

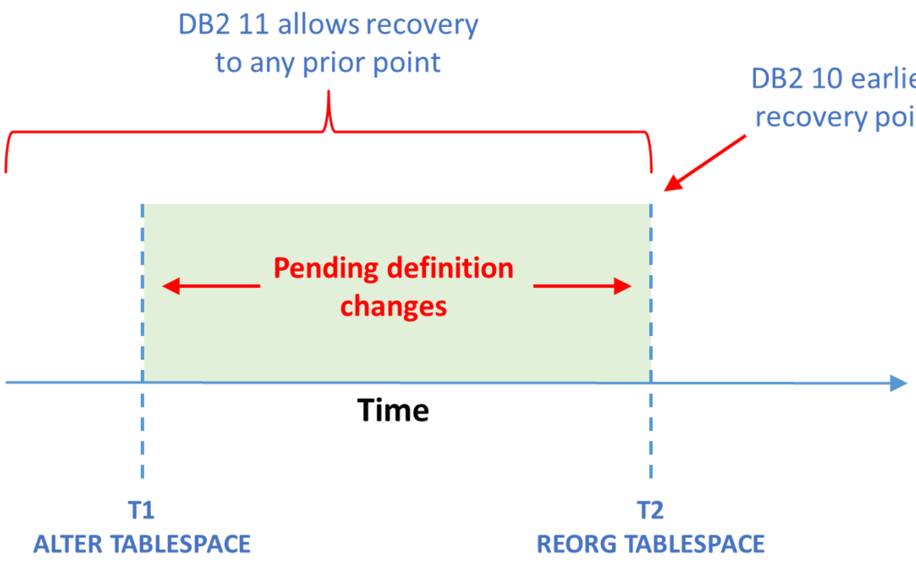

DB2 11 allows recovery to any prior point
DB2 10 earliest recovery point
Pending definition changes
Time
T1
ALTER TABLESPACE
T2
REORG TABLESPACE

Figure 5 - Recovery to PIT where pending definition changes exist

This is a significant enhancement, but there are some important considerations and restrictions to be aware of:

- Only UTS Partition by Range (PBR), LOB and XML table spaces are supported. Recovery of other tablespace types and indexes are subject to the same restrictions that existed in DB2 10.
- No clone tables may exist when RECOVER is initiated.
- The entire table space must be recovered to the prior PIT, individual partitions are not supported.
- The RECOVER VERIFYSET option (which specifies whether the RECOVER utility verifies that all related objects that are required for a point-in-time recovery are included in the RECOVER control statement) is overridden to YES, even if NO is specified.
- Upon completion of a recovery operation prior to the materialising REORG, the entire table space will be placed in a hard REORG-pending (REPORP) status and must be immediately reorganised (SHRLEVEL REFERENCE) before any data can be accessed. Any attempt to execute DML, DDL or most other utility processes[3] before running the REORG will fail. This may require additional steps to be added to standard recovery processes, and will extend the elapsed time of any recovery process.
- If the pending changes were on a LOB table space, the REORG must first be run against the LOB (auxiliary) table space and then a separate REORG is required against the associated base tablespace.
- Only the data is recovered to the previous point in time, the schema remains at the most recent version. This means that this technique cannot be used to “back out” pending schema changes. Using the example in Figure 5 above, if an ALTER TABLESPACE….DROP COLUMN was executed at T1 and a subsequently performed to a PIT between T1 and T2, the dropped column would not be present in the recovered table.

[3] REPAIR DBD and REPORT RECOVERY and RECOVER to same PIT are allowed.

In summary, this feature reduces the operational impact of allowing pending definition changes to persist until a convenient time presents itself for them to be materialised via a REORG. However, it is still advisable to materialise all pending changes via a REORG as soon as possible in order to avoid some of the issues above.

DROP PENDING CHANGES

DB2 10 for z/OS makes it possible to remove pending schema changes to a tables pace via the ALTER TABLESPACE…DROP PENDING CHANGES clause. This effectively "backed out" the schema change responsible for creating the pending changes, but left the table space in Advisory REORG Pending (AREOR) status even though there were no changes left to materialise.

DB2 11 does a better job of handling this situation and removes the AREOR status if pending changes are dropped.

BIND/REBIND ENHANCEMENTS

The advantages of REBIND are many. The opportunity to obtain better performance is, probably, the most important. There is a runtime overhead involved on running a package bound in a previous version of DB2. To obtain the CPU and memory advantages of a new DB2 release, and the benefits that may come with software maintenance requires REBIND. And you have to REBIND to get the benefits of the DB2 optimizer and of updated statistics in the form of a better access path. Some system restrictions may impose the need for REBIND. For example, only packages bound after or in DB2 9 can be used with DB2 11.

Nevertheless, the advantages and the need for REBIND have to be balanced with the risks of access path degradation. After so many year of DB2 evolution, ***to REBIND or not to REBIND*** is still the ubiquitous question. In consequence, many organizations have adopted a conservative approach and avoid REBIND as much as possible. Unfortunately, this policy often leads to missed opportunities in getting the most out of DB2 for z/OS.

DB2 has been adding features that help users to BIND and to REBIND in a safe way.

For example, DB2 9 introduced PLAN MANAGEMENT. It help users to "fallback" to a previous instance of a package when an access path change caused a performance regression. DB2 10 enhances PLAN MANAGEMENT support with the addition of new DB2 catalog tables and new BIND/REBIND options. DB2 11 continues this trend, and the following sections describe some of the most relevant changes in this area.

Some DB2 11 innovations relevant to this document are:

- Further PLAN MANAGEMENT capabilities with the introduction of APREUSE(WARN)
- BIND and DDL break-in option on persistent DB2 threads with RELEASE(DEALLOCATE) packages
- The option to specify the package compatibility level behaviour for static SQL (for more detailed, please refer to the SQL Compatibility Feature section on page 32)

FURTHER PLAN MANAGEMENT CAPABILITIES WITH THE INTRODUCTION OF APREUSE(WARN)

As a later addition to its General Availability announcement, DB2 10 introduced the APREUSE BIND/REBIND option. This option specifies to DB2 to try, or not, to reuse the prior access path for a SQL statement during BIND or REBIND of a package.

DB2 10 allows 2 different values for APREUSE:

- APREUSE(NONE): DB2 does not try to reuse prior access paths for statements in the package. This is the default value

- APREUSE(ERROR): DB2 tries to reuse the prior access paths for SQL statements in the package. If statements in the package cannot reuse the prior access path, the bind operation for the package fails and no new package is created. Processing continues for the next package.

You can think about this as having DB2 automatically using optimization HINTS. As such, there is no guarantee of success for every case. For example, if an index that is needed for applying a previous access path is missing, access path reuse will not work.

Nevertheless, APREUSE provides an effective way for packages to benefit from the DB2 10 runtime and virtual storage constraint relief while keeping a well-known access path. APREUSE exploits the information stored in the Explain Data Block (EDB) that was introduced in DB2 9, so only packages bound in DB2 9, or later, can take advantage of this feature.

A limitation with APREUSE(ERROR) is that all candidate SQLs must succeed in reusing their prior access path or the entire package REBIND fails. Customers have been looking for a solution that is would allow more packages to succeed reuse despite some SQL failures within a package.

DB2 11 delivers an additional option for APREUSE – WARN. With APREUSE(WARN), DB2 tries to reuse the prior access paths for SQL statements in the package, but the bind or rebind is not prevented when they cannot be reused. Instead, DB2 generates a new access path for that SQL statement. APREUSE(ERROR) therefore can be considered as operating on a package boundary, and APREUSE(WARN) operating on a statement boundary.

Now users can mass REBIND a DB2 environment and maximize the number statements that can reuse their prior access path. As with HINTS, it is normal to observe a small percentage of the packages failing. Using APREUSE(WARN) you get a new package and the notification that reuse was not possible. Not being able to reuse an access path is not necessarily an issue, because it could happen that the new access path is actually better than the previous one. An exploration of the PLAN_TABLE can help to decide between the options of allowing the package to run with the new access path, or to switch back to a previous package. This last possibility requires the use of the PLANMGMT bind option.

DB2 11 improves the diagnosis information made available for cases in which reuse cannot be applied. When APREUSE is used in combination with EXPLAIN(YES) or EXPLAIN(ONLY), DB2 populates PLAN_TABLE data with access path information.

In DB2 11, the PLAN_TABLE accurately describes the valid new access path even in case of reuse failure. This was not always the case with DB2 10. This improvement applies regardless of whether APREUSE(ERROR) or APREUSE(WARN) is specified. Although with APREUSE(ERROR), failure information is only reported to the PLAN_TABLE.REMARKS column with EXPLAIN(ONLY). Given that a valid access is now generated and recorded in the PLAN_TABLE upon an APREUSE failure (with ERROR or WARN), then the exact reason for the APREUSE failure may not be easy to determine.

With EXPLAIN(YES) or EXPLAIN(ONLY), successful reuse is reported by setting PLAN_TABLE.HINT_USED='APREUSE' which is the same as in DB2 10.

This example shows the DB2 11 command syntax changes:

```
>>---BIND-PACKAGE----+----------------------------------+-------------->
                     |               '---NO---'         |
                     |               '--NONE--'         |
                     '----APREUSE-(--+--WARN--+--)-----'
                                     '--ERROR-'
```

```
>---REBIND-PACKAGE---+----------------------------------+-------------->
                     |              '---NO---'          |
                     |              '--NONE--'          |
                     '----APREUSE-(--+--WARN--+--)-----'
                                    '--ERROR-'

>---REBIND-TRIGGER-PACKAGE---+----------------------------------+------>
                             |              '---NO---'          |
                             |              '--NONE--'          |
                             '----APREUSE-(--+--WARN--+--)-----'
                                            '--ERROR-'
```

SUMMARY

DB2 11 enhances its plan management functionalities with the option APREUSE(WARN). Users can now plan for safe BIND/REBIND strategies with a more usable access path reuse functionality.

BIND AND DDL BREAK-IN OPTION ON PERSISTENT DB2 THREADS WITH RELEASE(DEALLOCATE) PACKAGES

A package running with the option RELEASE(DEALLOCATE) will release resources only when the program terminates. In contraposition, a package running with RELEASE(COMMIT) releases resources at COMMIT, *unless* the execution characteristics of the package make resource to be kept across COMMITs. This may happen when the package uses CURSORS WITH HOLD, Declared Temporary Tables that are not explicitly dropped, or when the package runs with KEEPDYNAMIC(YES), for example.

RELEASE(DEALLOCATE) has no effect on most dynamic SQL statements, which use RELEASE(COMMIT), except in the following cases:

- When a package runs with RELEASE(DEALLOCATE) and KEEPDYNAMIC(YES), and DB2 is running with the Dynamic Statement Cache enabled (CACHEDYN = YES), the RELEASE(DEALLOCATE) option is honoured for dynamic SELECT, INSERT, UPDATE, and DELETE statements
- During the execution of a package bound with RELEASE(DEALLOCATE), prepared INSERT, UPDATE, DELETE, and MERGE dynamic statements that reference declared temporary tables are kept across COMMIT unless the table was defined with the ON COMMIT DROP TABLE option.

RELEASE(DEALLOCATE) may increase the memory footprint of a package because more resources have to remain allocated with the thread. With the memory improvements provided by DB2 10, where most of the thread storage is moved above the bar, this situation is usually not a problem anymore. Users are starting to consider and to use RELEASE(DEALLOCATE) more frequently than in previous versions of DB2.

The practical application of RELEASE(DEALLOCATE) shows that this option can provide a substantial performance advantage for some types of applications, like Online Transaction Processing systems characterized by and intensive workload composed of short SQL queries.

Nevertheless, a fundamental problem remains till before DB2 11: some administrative operations like BIND/REBIND, and some DDL changes, could not be executed if a package running with RELEASE(DEALLOCATE) is persistently retaining locks on the target objects. Database administrators usually have to identify the offending DB2 threads and terminate them before being able to apply the desired changes

With DB2 10, High Performance DBATs make available the RELEASE(DEALLOCATE) option to distributed access to DB2. To circumvent the eventual concurrency problems, database administrators can issue the MODIFY DDF

PKGREL(COMMIT) DB2 command to temporarily override the BIND/REBIND option and to be able to break-in otherwise persistent threads. In DB2 10 there is no equivalent facility for non-distributed applications.

DB2 11 introduces a new subsystem parameter: PKGREL_COMMIT.

PKGREL_COMMIT defines whether BIND, REBIND (including auto rebind), and certain DDL changes are permitted on a package that is bound as RELEASE(DEALLOCATE) while the package is active and allocated by DB2.

The allowed values for PKGREL_COMMIT are:

- NO: Do not permit these operations
- YES: Permit the operations. They will take place when locks are released at the next COMMIT or ROLLBACK point

The default value in DB2 11 is YES.

INSERT PERFORMANCE IMPROVEMENTS

As information processing continues to evolve and cost of CPU and mass storage decreases, we find more and more organizations storing vast quantities of data in DB2. In many cases this data is arriving at very high rates. I have seen single table insert rates as high as 39 million per day and single table insert rates as high as 13,000 inserts per second. I've also seen system-wide insert rates as high as 17,000 per second. Special table designs are needed for tables to accept inserts at such high rates. So, DB2 customers use such features as APPEND ONLY, MEMBER CLUSTER, and larger index page sizes to improve the insert rates for some of their tables. Using these design options, along with some clever application design, these insert rates and even high rates are possible. What becomes obvious once these high rates are achieved is that there are certain system factors that may prevent the rates from getting even higher. Once the table and table space bottlenecks are overcome the prominent limiting factors to even higher insert rates are logging and latching.

DB2 11 for z/OS provides relief to some of the major system level bottlenecks that are associated with high rates of inserts. These changes are all internal and require no intervention on behalf of the user. One of the improvements is in the reduction of latching class 6, 14, and 19 related to INSERT processing. Another improvement is related to the new 10-byte LRSN value (please see Extended log record addressing on page 7). In previous versions of DB2, INSERT related processing had to spin in order to wait for a unique LRSN value for a new row being inserted into a specific page in a table space. Given that LRSN precision was only to 16 microseconds this limited the number of rows that could be inserted to a single page to 1 every 16 microseconds. The use of a 10-byte LRSN breaks that barrier. There are additional internal enhancements related to the avoidance of page fixing and freeing when writing to group buffer pools in a data sharing environment. In addition to all of this there are various other performance improvements that can contribute to faster inserts including more efficient compression and more efficient index page split processing.

Almost all of these improvements come without cost or effort by the user to implement with the exception of the extended LRSN format. In order to implement extended LRSN usage for a table space the bootstrap datasets will have to be converted and a REORG will be required for existing table spaces to take advantage of the new LRSN format. In addition, the bootstrap datasets will first have to be reformatted to enable the extended LRSN.

SQL COMPATIBILITY FEATURE

Many new releases of DB2 introduce enhancements or new features that require application and/or SQL code to be changed. These include additional SQL reserved words, changes to DB2 behaviour or processing and even changes to SQL return codes. Although IBM tries to minimise these “incompatible changes”, they cannot always be avoided – they may be required in order to ensure that DB2 adheres to evolving SQL standards, to support new functionality, or perhaps to address an earlier defect in the DB2 code.

Incompatible changes are listed in the relevant section of the DB2 Installation and Migration Guide, and a major part of planning for a new release is to analyse their impact and arrange for the necessary changes to be made to application code so it will continue to work as designed under the new release.

This situation poses several challenges for DB2 customers:

- Analysis of the impact of incompatible changes can be difficult, time consuming and error-prone. Missing one or more of the required changes may result in application outages when DB2 is upgraded (or even worse, the application may continue to work but return unexpected results).
- Finding the necessary resource to undertake any necessary remedial work (and scheduling the associated change slots) can be expensive and require significant elapsed time. All of the changes within a given subsystem or data sharing group must be completed before the upgrade can commence, so a lack of resource within a single application team could impact the upgrade schedule for the entire environment.

In order to address these issues, IBM has introduced some new capabilities in DB2 11 for z/OS that remove the hard dependency on all remedial work being conducted prior to a version upgrade, and allow the impact of incompatible changes to be more easily assessed. We’ll look at each of these separately in the sections that follow.

DEFER REMEDIAL WORK FOR INCOMPATIBLE CHANGES

IBM laid much of the groundwork for easier management of incompatible changes when fixing a previous release incompatibility issue within DB2 10 for z/OS. APAR PM29124 describes a problem caused by a DB2 10 release incompatibility involving character representations of decimal numbers. The fix for that APAR introduced a new ZPARM (BIF_INCOMPATIBILITY) to force DB2 to adopt the old (DB2 9) behaviour as an alternative to having to change all affected applications.

The concept has been extended in DB2 11 to cover all incompatible SQL DML and XML changes, and allow the DBA or developer to request that DB2 behaves the same as it did for a prior release at the individual plan/package level (or statement level for dynamic SQL). This removes the requirement for the incompatible change remedial work to be conducted prior to a version upgrade: from DB2 11 onwards, a new DB2 release can be implemented without any remedial work being completed, and DBAs and developers are free to address that work at a later date and in a more manageable, staged fashion (e.g. as part of a scheduled application release).

The intention of this feature is to allow more manageable implementation of remedial work not to defer it work indefinitely, so this capability is limited in the number of previously supported releases. In the initial DB2 11 implementation of this feature, only DB2 10 compatibility is provided but beyond that two previous releases will be allowed. This means that the release following DB2 11 will support both DB2 10 and DB2 11 compatibility, thereby allowing plenty of time for any remedial work to be undertaken.

New DSNZPARMs, BIND options and special registers have been added to support DB2 11 application compatibility, as described below.

- The new APPLCOMPAT BIND/REBIND option tells DB2 which release behaviour to use for the package being bound. This can only be set to “V10R1” in DB2 11 Conversion Mode, thereby enforcing DB2 10 compatibility behaviour for all packages while in CM. Any packages bound prior to V10 are treated as if they had been bound with APPLCOMPAT(V10R1), and remember that any packages bound prior to DB2 9 will be invalidated and will go through automatic rebind processing. Once in New Function Mode, individual packages can be rebound with APPLCOMPAT(V11R1) as required (i.e. when any remedial work has been completed). Please see BIND/REBIND Enhancements on page 29 for further information on the DB2 11 enhancements in this area.

- ·The CURRENT APPLICATION COMPATIBILITY special register allows the same information to be supplied for dynamic SQL, with the default being taken from the APPLCOMPAT value for the associated package.
- The APPLCOMPAT DSNZPARM specifies the default APPLCOMPAT BIND option if it's not explicitly specified in the BIND command.

IMPACT ANALYSIS FOR INCOMPATIBLE CHANGES

Being able to defer remedial work beyond the release boundary is a great benefit, but once you've bought yourself some time to make the changes how do you go about knowing exactly which applications need to be changed?

Again, APAR PM29124 for DB2 10 laid the foundations. It introduced a new IFCID366 trace record, which could be written every time DB2 detected the execution of a plan or package using the BIF_INCOMPATIBILITY workaround. Simple analysis of the trace records then allowed the DBA or Sysprog to obtain a list of plans/packages that needed to be changed.

IFCID366 has been extended in DB2 11 to report on all of the incompatible SQL and XML changes introduced in the new release, and works with both static and dynamic SQL. A new trace record, IFCID376, has also been introduced. This is essentially a "rolled up" version of IFCID366, with one record written for each unique static or dynamic SQL statement (as opposed to one record per execution with IFCID366).

Between them, these two IFCIDs provide a good means of determining which applications may need remedial work following the upgrade – any DB2 performance monitor will be able to produce a report listing the plans/packages requiring investigation.

SUMMARY

The DB2 11 Application Compatibility feature addresses many of the issues associated with incompatible changes in each new DB2 release. However, its restrictions and limitations must also be recognised. IFCID366/376 will only be produced when a relevant back-level SQL/XML statement is executed, so care must be taken to ensure that infrequently run processes (e.g. year-end) are included in any analysis. Some categories of incompatible change (e.g. changes to SQLCODE processing) will still require manual analysis. Most importantly, the Application Compatibility feature is not a means to permanently defer the remedial work needed to address incompatible changes: it provides a means to postpone such changes so they can be scheduled at a convenient time without impacting the overall DB2 upgrade schedule. Customers who allow the remedial work to stack up for two DB2 releases will find themselves in exactly the same position as they are today – forced to make changes to their applications prior to the next release.

Overall, this capability should be a huge benefit to customers struggling to line up the necessary application development/DBA resource to address any incompatible changes prior to a new release of DB2 being implemented. It breaks the hard dependency on performing all remedial work prior to an upgrade, and provides valuable tools to assist with the identification of that work.

XML ENHANCEMENTS

Way back we used IMS DB. Everything was hierarchical as the roots were constructed to handle massive BOM (Bill of Material) needed for the Saturn moon rocket and the Apollo project. We lived happily with that and no one knew better until the arrival of the relational model. From that day, our life has changed.

No more complex hierarchies and complicated program logic to handle data. SQL became the king. Handling SSAs and playing around with how to access the data disappeared in a blink. Some complained, some objected, some claimed that IMS is better / faster / … pick your choice. However the vast majority voted with their feet (actually – hands) and moved over to SQL with the notion of never looking back.

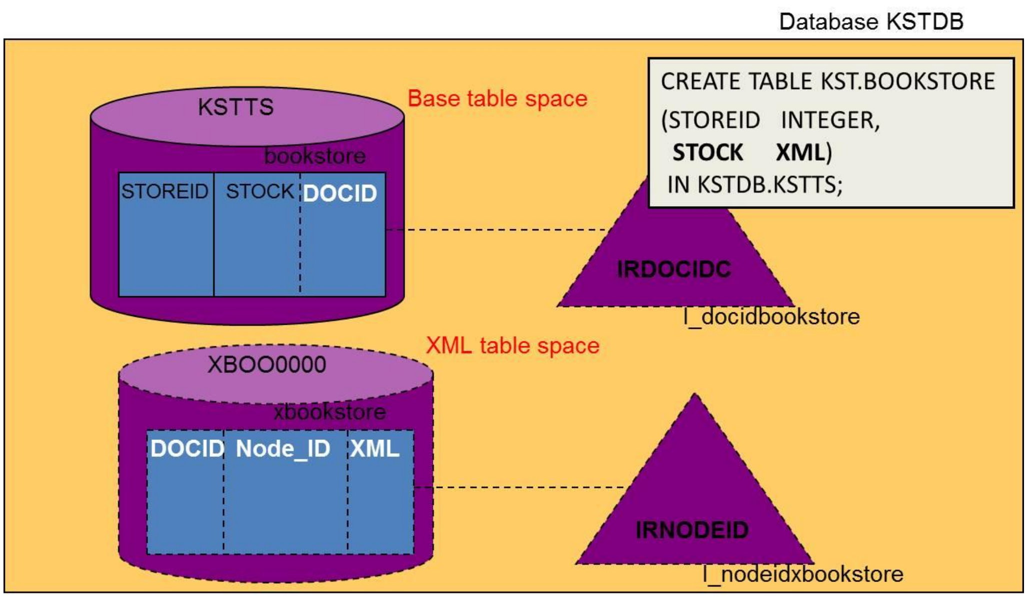
DB2, being IBM's flagship in the relational DBMS market has come a long way since its' debut. With experience we had some new design ideas which were a convoluted hierarchy. We found out that some designs really wanted a touch of hierarchy to work better. However, we preferred to build some workarounds and did not want to look back (to IMS).

More experience and demands along the way caused us to look at XML as not only XML for the internet world but as a way to solve our really needed demands from a hierarchical view of our data. Using XML for hierarchical structures within DB2 has become a way of life - not the exception.

DB2 11 brings many improvements in the XML area. The main improvement is the possibility to use XQuery. This is such a big improvement; it has been retrofitted to DB2 10 (UK72208 & UK73139). In the world of data exchange, XML has become the global standard, any data professional should know the basics of XML processing and how it interacts with their database system. A review of the possibilities, combined with the new features might be useful.

STORING XML IN DB2 FOR Z/OS

With the arrival of DB2 9 for z/OS, it became possible to store XML in a native format as a column in a DB2 table. Before this possibility, the DBA had to resort to using extenders or to storing the XML document in a LOB column. This last solution is still a valid option today, especially if simply storing and retrieving the XML document is the only concern. However, if a full use of the XML capabilities is desired then the XML data type should be utilized. Figure 6 shows what happens when a table named "bookstore" is created, containing column "stock" of the XML data type.



```
CREATE TABLE KST.BOOKSTORE
(STOREID   INTEGER,
 STOCK     XML)
 IN KSTDB.KSTTS;
```

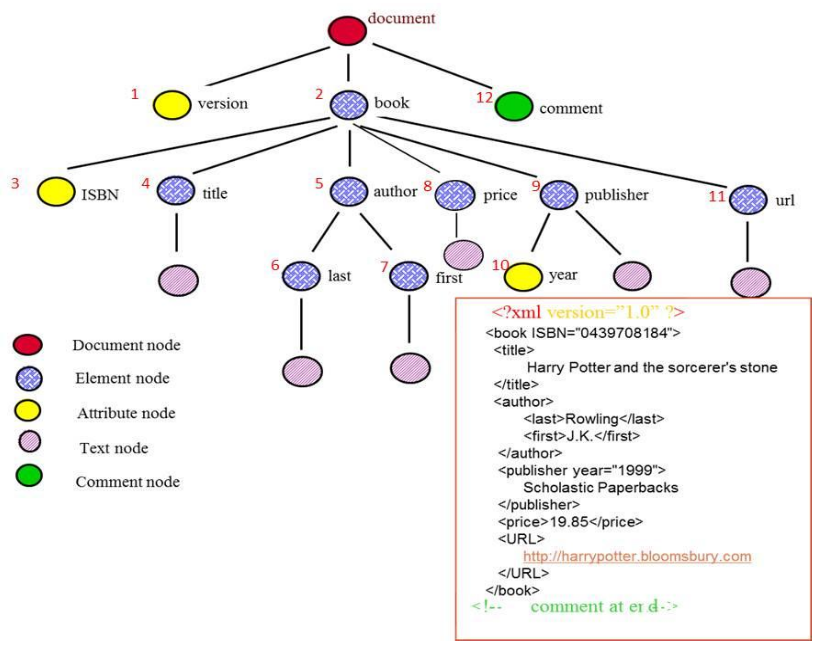
Figure 6 – Creating an XML Object in DB2

Assuming an explicitly created database (KSTDB) and table space (KSTTS), a table (bookstore) will be created with the two specified columns (“storeid” and ”stock”). Notice however DB2 will implicitly create an extra column “docid” that is used to link the base table with the implicitly created XML table (xbookstore). The XML table resides in an implicitly created tablespaceXBOO0000. On top of this DB2 also provides two indexes, one on the base table and one on the XML table. This structure is very similar to the physical structure used for LOBs.

The implicitly created table space is either a partition by growth or a partition by range table space, depending on the base table space definition. Optimally, there are two considerations to make when choosing the correct base table space type. First, the number of records a base table space partition can hold depends directly on the size and number of XML documents to be held in the corresponding XML table space. E.g. average 4K XML doc size, 32GB partition can roughly store 8M documents, let’s say 6M to be safe. If a base row is 100 bytes, 6M rows means only 600M per partition in the base table. The XML table space inherits the compress attribute from the parent, which can lead to significant space reduction if the “preserve white space” option is in effect. The second consideration is that the base table space is best created as a universal table space. This will be discussed further on in this chapter.

NATIVE XML SUPPORT

By choosing the XML data type as part of the table design, DB2 offers native XML support. This means that inside the XML column (“stock”) there is a hierarchal structured document. This allows the retrieval of the full or partial document. It also makes filtering on the XML elements and attributes possible.



```
<?xml version="1.0" ?>
<book ISBN="0439708184">
 <title>
       Harry Potter and the sorcerer's stone
 </title>
 <author>
      <last>Rowling</last>
      <first>J.K.</first>
  </author>
  <publisher year="1999">
      Scholastic Paperbacks
  </publisher>
  <price>19.85</price>
  <URL>
      http://harrypotter.bloomsbury.com
  </URL>
</book>
<!--    comment at end-->
```

Figure 7 - XML Example

Figure 7 displays the logical layout of the stock column and the order in which (numbers 1 through 12) the nodes will be visited. Besides the overview, an example XML document of “stock” is shown, containing the information about a book.

The book example shows that every element is enclosed in an opening <tag> and a closing </tag>. An XML document has to follow a number of rules in order to pass the XML parser. When a document is parsable, it’s called “well formed”. DB2 will verify XML documents, and guarantees that every XML document stored in its tables is well formed.

Merely being well formed might not be enough for an XML document. Some documents might have to abide by some external logical rules. These rules could dictate the minimum number of occurrences for an element, force data type checking of elements, and more. These rules are either defined in a Document Type Definition (DTD) or in an XML Schema Definition (XSD). The method of using DTD is rapidly going out of fashion. XML schema is currently used, as it is more powerful, more flexible and less cryptic.

DB2 for z/OS supports the use of XML schema, and allows users to register and use it to validate XML documents in a table. Depending of the version, this schema validation will have to be explicitly coded at every insert (version 9 and up) or the possibility is given to implicitly check the document based on the XSD definition at a column level (version 10 and up).

```
INSERT INTO KST.BOOKSTORE
VALUES (12, SYSIBM.DSN_XMLVALIDATE(:xmldoc, ’SYSXSR.KURT’))
```

```
CREATE TABLE KST.BOOKSTORE
(STOREID   INTEGER,
 STOCK    XML (XMLSCHEMA SYSXSR.KURT))
IN KSTDB.KSTTS
```

Figure 8 - XML Schema Validation

Figure 8 shows both techniques (explicit and implicit) of XML schema validation in DB2 for z/OS.

Evidently, validating an XML document is significantly more CPU intensive than merely checking if it is well formed. DB2 10 needed to serialize the binary XML into string XML, in order to validate.

In order to check whether or not an XML document has been validated, a built in scalar function XMLXSROBJECTID exists. For example, XMLXSROBJECTID(STOCK)=0, returns the non-validated XML documents.

Fortunately in DB2 11 brings some nice improvements regarding the validation of XML documents. Firstly, validating XML documents is ZIIP off-loadable. Secondly, DB2 11 validates binary XML directly, both during insert and load. This can lead to a significant performance benefit.

To create an XML schema, you must introduce them into DB2. This process uses two stored procedures (included in DB2) XSR_register and XSR_complete. All schemas must be registered under the “schema SYSXSR”.

Figure 9 below shows an extract out an XML schema definition; it shows the rules to be followed for the “publisher” element in the bookstore example.

```
<xs:element name="publisher" maxOccurs="1">
        <xs:complexType>
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="year" type="xs:integer" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

Figure 9 – Example of an XML Schema Definition

XPATH NAVIGATION

XPath is a language that describes a way to locate nodes in XML documents by using a syntax based on a path through the document's hierarchical structure. It is position based navigation. A slash ("/") will navigate the user from its current position to the child of the current position. A double slash ("//") will navigate the user from its current position towards a descendant (child, grandchild, great grandchild etc.)

In the earlier bookstore example (see Figure 7 above) navigating down to last name of the author starting at the root would involve following the path: /book/author/last. At this point, it's important to indicate that XPath expressions (and XML values in general) are case sensitive, so /BOOK/AUTHOR/LAST would not be a correct navigation within the document. Should an attribute node be searched the @-sign would be used. In the earlier example navigating to the publishing year would be translated in XPath as: /book/publisher/@year.

Frequently not all element nodes will need to be returned to an application. In that case XPath provides the possibility to code conditions. Conditions will always be coded between square brackets [ ], so looking for all book titles published in "1999" would result in /book[publisher/@year="1999"]/title

XPath expressions are supported in DB2 for z/OS to retrieve values from XML columns using the XMLQuery function. It is important to note that XMLQUERY will filter the result coming out of an XML document, but it will not filter the number of rows being returned from the table. If bookstore has 1500 rows and only one store would have the book, DB2 will return 1500 rows, only 1 row contains the book, all other rows are empty sequences which are the analogue of the SQL null value. The XMLEXISTS predicate enables predicates on XML based conditions and therefore filters the amount of rows being returned. An example can be seen in Figure 10.

```
SELECT XMLQUERY('/book/[publisher/@year="1999"]/title' PASSING STOCK )
FROM KST.BOOKSTORE
WHERE XMLEXISTS('$x/book[@ISBN="0440239494"]'
  passing STOCK as "x")
```

Figure 10 – Example of an XML Query

This shows an example of an XMLQuery, returning all the titles of all the books in all the bookstores published in 1999 if the ISBN number = 0440239494.

Indexes can be defined to support XMLEXISTS predicates. As with any index there is an overhead for maintaining the index during data manipulation language (DML) statement execution. With the understanding that this DB2 has to scan an XML document at insert to determine the index value, this overhead can be significant.

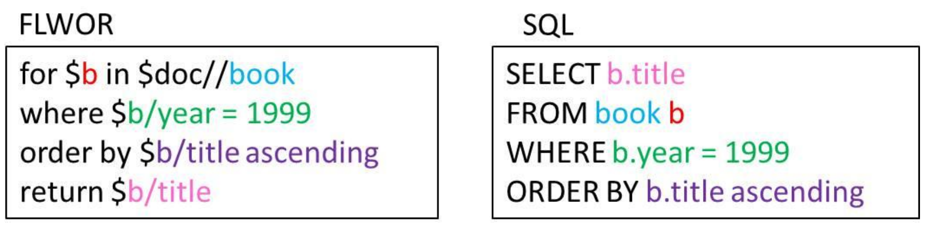
XQUERY NAVIGATION

XQuery is new to DB2 11 and retrofitted to DB2 10. XQuery is primarily devised as a query language for data stored in XML format. It is based on XPath expressions but allows for more flexibility and expressiveness.

Talking about XQuery means talking about FLWOR (read flower) –expressions. FLWOR stands for:

- "For", defining the source.
- "Let", allowing variable declaration and value assigning.
- "Where", specifying search conditions
- "Order by", sorting the result
- "Return", defining the output of the expression.

This may seem abstract, but it actually is closer to SQL then one might think. Most DBAs quickly understand the similarities.



FLWOR

```
for $b in $doc//book
where $b/year = 1999
order by $b/title ascending
return $b/title
```

SQL

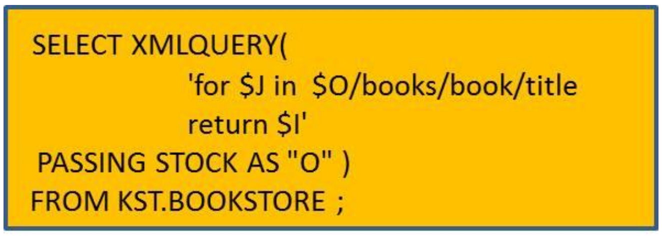
```
SELECT b.title
FROM book b
WHERE b.year = 1999
ORDER BY b.title ascending
```

Figure 11 – Comparison of FLWOR and SQL Syntax

The example in Figure 11 is a basic FLWOR expression. Through the use of IF-ELSE-THEN constructions, mathematical expression and result transformation (to name a few), the use of XQuery is vast and flexible.

For mainframe DBAs it is important to remember, XQuery statements are case sensitive. This means everything, not just the path expression. Review the example in Figure 11 and notice that for example the “for” keyword is in lower case. When coded “FOR” in uppercase, this causes an SQL error -16002.

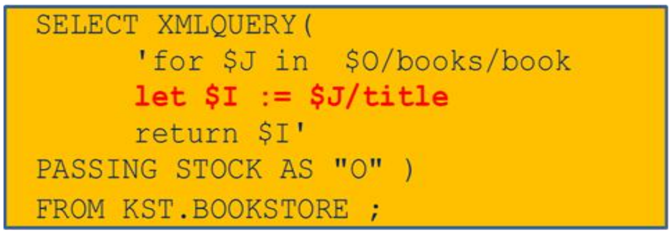
In Figure 12 a basic FLWOR expression is given on our bookstore example once as straight forward as can be. This is a rewrite of the XPath expression, just to illustrate the FLWOR syntax.



```
SELECT XMLQUERY(
             'for $J in  $O/books/book/title
             return $I'
 PASSING STOCK AS "O" )
FROM KST.BOOKSTORE ;
```

Figure 12 – Simple FLWOR Example

Figure 13 shows the same but using a let-clause:



```
SELECT XMLQUERY(
       'for $J in  $O/books/book
       let $I := $J/title
       return $I'
PASSING STOCK AS "O" )
FROM KST.BOOKSTORE ;
```

Figure 13 –FLWOR Example using LET

Of course more complex data modification or calculations are possible, which make XQuery the more powerful XML query option. A slightly more advanced but still very simple example is shown in Figure 14. This is still very basic compared to what can be done in XQuery, but it might give an idea of what can be accomplished.

```
SELECT XMLSERIALIZE(XMLQUERY(
      'for $J in  $O/books/book
       let $I := $J/title,
           $A := $J/awards/award/name,
           $Q := fn:count($A)
      where $Q > 0
  return  <book> {$I}
           <awards> {$Q}
             <awardname> {fn:data($A)} </awardname>
           </awards>
          </book>'

      PASSING STOCK AS "O" ) AS CLOB)
  FROM KST.BOOKSTORE
WHERE STOREID = 1
```

Figure 14 –More Complex FLWOR Example

Another example of the power of FLWOR is to iterate through two data sources and join them. Imagine bookstore had a second XML column "review" then using FLWOR, you can join those two XML documents.

```
SELECT XMQUERY(
               'for $i in $o//book,
                    $j in $r/reviews/book
               where $i/@ISBN = $j/@ISBN
               return ...'
passing stock as "o", review as "r")
FROM  ...
```

Figure 15 –Joining XML Documents

UPDATING XML DATA

DB2 11 provides the XMLMODIFY function to update individual XML-nodes rather than the entire document. In order for this XMLmodify function to work, the base table must reside in a universal table space.

The XMLmodify function is based on a basic replace logic, where the path of the node needing to be changed is replaced with another (calculated or not) value. Imagine a certain book in bookstore receiving awards and becoming very popular. A certain store (imagine store 3) might want to increase the price for this book by 15%. This means that the XML document in the "stock" column would have to be updated.

```
UPDATE KST.BOOKSTORE1 SET STOCK = XMLMODIFY
  ('replace value of node /books/book[title="FEELS LIKE HOME"]/price
    with /books/book[title="FEELS LIKE HOME"]/price*1.15')
where storeid = 3
```

Figure 16 –Updating an XML Document

Figure 16 shows the price of the book "feels like home" being increased by 15% for store 3 in the XML column stock.

XML CONSTRUCTS

A common use case is to construct data for an XML column, using construct functions like XMLELEMENT. The idea is to build an xml document based on data currently already residing in you relational database design. We've had this possibility in previous versions of DB2.

```
INSERT INTO KST.CUSTOMER
VALUES('123', (SELECT XMLDOCUMENT (
                        XMLELEMENT (NAME "Emp"
                                                    ,XMLATTRIBUTES(E.EMPNO as "ID")
                                                    ,E.LASTNAME),
                        XMLCOMMENT ('This is just a simple example')
                                          )
                        FROM KST.EMP E
                        WHERE E.EMPNO = '000010'
              )
          ) ;
```

Figure 17 – How an XML document can be Built and Inserted, Based on Relational Data

However in DB2 11 we can use the XQuery constructor, to build a similar XML document.

```
insert into KST.CUSTOMER
values (123,    (SELECT XMLQUERY( 'document
                    {<Emp ID="{$EMPNO}">{$LASTNAME}</Emp>}'
                passing E.EMPNO as EMPNO, E.LASTNAME   as
                LASTNAME)
                From kst.emp e
                where e.empno = '000010' ) );
```

Figure 18 –Using an XQuery constructor to build XML data

In both examples an insert is done in the two-column (id and cust_info) customer table. The second column is an XML column. The values of this column are built from data in the emp table.

In order to create a document node (needed for every XML document) up until version 10, an explicit call of the XMLDOCUMENT function is needed. Failure to code this function results in a -20345 SQL code.

DB2 11, inserting or updating XML documents in a table, no longer requires an explicit call of the XMLDOCUMENT function. Should this function be omitted in DB2 11, an implicit document node will be added by the DB2 engine.

With XQuery support in DB2 11, a broader support for XML is available in DB2. It allows for more complex XML use cases. The importance of XML data transfer is still increasing.

UTILITY ENHANCEMENTS

Each new release of DB2 brings changes to the utilities suite, and new release DB2 11 is no exception. The importance of these changes can have a big impact on the work of a DBA. The changes and new options should be well understood and choices have to be made for your environment. The purpose of this article is not to provide a full and detailed list of those improvements, nor to provide full syntax. The idea is to introduce some of those features.

THE REORG UTILITY

Each new release of DB2 brings changes to the utilities suite, and new release DB2 11 is no exception

AUTOMATED MAPPING TABLES

Have you ever tired of managing Mapping Tables for Online Reorgs? Have you ever received one of the Messages below?

```
DSNU366I - REORG MAPPING TABLE HAS WRONG COLUMNS

DSNU368I - REORG MAPPING TABLE HAS WRONG INDEX

DSNU371I - REORG MAPPING TABLE'S TABLE SPACE MUST BE SEGMENTED
```

DSNU372I - CANNOT REORGANIZE REORG MAPPING TABLE'S TABLE SPACE

Each REORG job uses its own mapping table and it cannot be shared by other concurrent REORGs. Let's say you have configured and tuned your operational environment for ten concurrent Reorg Jobs. If you want to increase the number of reorg jobs, you need to manually create additional mapping tables (besides other things). There is also a scalability constraint, with mapping table indexes being subject to a 64GB maximum size, thereby limiting the number of rows that can be REORGed. In V11, RBA/LRSN size is also changing (please see Extended Log record addressing on page 7), so the DDL for all of your existing mapping tables must change too.

All of these issues can be resolved with the automation of mapping table management in DB2 11 for z/OS, which is perhaps the most eye-catching new enhancement with the new REORG utility. This means that, should this be used in your shop, DB2 will take care of the allocation and removal of the mapping table during a SHRLEVEL CHANGE reorganization. This doesn’t mean that you cannot specify your own mapping table anymore; it’s just that you now have the option to let DB2 handle it. Should you choose to keep on working with your own mapping table, you should be aware of the new layout of the mapping table in DB2 11 due to the change in RBA/LRSN length. When a reference is made to an invalid mapping table (non-existing or wrong format) DB2 will allocate the necessary mapping table.

Automated Mapping Tables will reside in a PBG tablespace and mapping index maximum size will be increased from 64GB to 16TB.

Here is how REORG will decide to create or use the existing mapping table:

1. If mapping table specified & correct format then honour specification

2. Else if specified but incorrect format then create new in same DB as original

2.5 keyword MAPPINGDATABASE overrides ZPARM / implicit database if specified

3. Else if not specified and ZPARM DB specified then create in ZPARM DB

4. Else create in implicit DB

5. DROP at end of REORG or end of last REORG if multiple REORGs in job step

The new ZPARM REORG_MAPPING_DATABASE, specifies the default database in which REORG TABLESPACE SHRLEVEL CHANGE should implicitly create the mapping table. If this parameter is set to a valid database, REORG will allocate the mapping table in that database. If this parameter is set to null, REORG will allocate the mapping table database in an implicitly-defined database

There is also a new REORG keyword called MAPPINGDATABASE. By using this, you can control the location of mapping table on REORG level rather than subsystem level.

REORG REBALANCE – ONLINE!

REORG REBALANCE is presented as a great enhancement in V8. It specifies that REORG TABLESPACE is to set new partition boundaries so that rows are evenly distributed across the reorganized partitions. If the columns that are used in defining the partition boundaries have many duplicate values within the data rows, even balancing is not always possible. As of V10, You can specify REBALANCE with SHRLEVEL NONE or SHRLEVEL REFERENCE. REBALANCE cannot be specified with SHRLEVEL CHANGE. REORG REBALANCE may fail if non-unique partitioning key and data skew due to inability to distribute data evenly across partitions

V11 provides support REORG SHRLEVEL CHANGE REBALANCE and improve resiliency with enhanced distribution algorithm & improved handling of empty partitions. If run out of data but still have parts to process, set new limitkey values for empty partitions.

PBG PARTITION CLEANUP

Another improvement in the reorg utility allows cleaning up empty partitions in partition by growth table spaces. Deleting empty partitions of PBG table spaces is a system wide choice and cannot be specified at a reorg statement. It requires a DSNZPARM change. The default behaviour is similar to previous versions of DB2: after REORGing, physical datasets won't be deleted unless the new DSNZPARM parameter REORG_DROP_PBG_PARTS is set to ENABLE

SORTDATA NO

DB2 11 extends the SORTDATA NO option for SHRLEVEL CHANGE reorgs. This keyword causes the data to be unloaded according to the clustering index, rather than unloading page by page and passing that data on to DFSORT to be sorted into the clustering sequence. Specifying SORTDATA NO can be very useful when the data is in or near perfect clustering order and the REORG utility is used for reclaiming free space of dropped tables. Another reason for running a REORG with SORTDATA NO can be a very large amount of data and insufficient amount of disk space to accommodate the sort. RECLUSTER NO in addition to SORTDATA NO will allow the fastest path by unload using a tablespace scan, and skipping the sort before reloading. This is useful when materializing schema changes where clustering order doesn't matter.

```
REORG DB.TS .... SORTDATA NO
```

IMPROVED SWITCH–PHASE PROCESSING

DB2 11 introduces an improved switch –phase processing during an online reorg. It will significantly reduce the switch phase of partition level reorgs and thus improving availability. Starting in DB2 11, DB2 will no longer allow new claims on target partitions while preparing to switch.   The partitions are drained one partition at a time while marking all partitions to avoid new claimers.

```
REORG DB.TS .... DRAIN WRITERS
```

This allowed readers to continue connecting to the table space being REORGed, while DB2 was in the last log iteration of a SHRLEVEL CHANGE REORG. The problem that could arise is, that one of these readers would take such a long time, that it would cause the REORG utility to timeout in the switch phase.

Starting in DB2 11 the default behaviour is, that of the option:

```
REORG DB.TS .... DRAIN ALL
```

This means that once DB2 starts its last log iteration phase, by default both readers and writers will be drained of the target partitions. . A long reader can still cause a timeout, but in most scenarios going directly to DRAIN

ALL ends up being faster and less prone for "undetected" deadlock conditions with various scenarios. This is the new default.

NEW SWITCHTIME PARAMETER

When talking about this last log iteration phase it is worth mentioning that there is a new option in place specifically designed to control the absolute last moment this last log iteration phase should start. It is specifically designed as a usability option to get around the use of MAXRO DEFER where one is trying to time the SWITCH data available outage with a batch window. SWITCHTIME identifies the beginning of the window so one can avoid the use of ALTER MAXRO, to complete the REORG.

```
REORG DB.TS .... SWITCHTIME
```

The switch time keyword specifies the time for the final log iteration of the log phase to begin. By default no value is provided, meaning that the last log iteration begins when needed in the reorg utility. Other options are either a hardcoded timestamp or a labelled expression (e.g. Current timestamp + 4 hours).

```
REORG DB.TS .... SWITCHTIME timestamp

REORG DB.TS .... SWITCHTIME current timestamp + 4 hours
```

The goal is to control the ending of the reorg utility. It is similar to the deadline option, however it is a more gentle way of steering the reorg utility towards its end. Reaching the deadline timestamp or labelled expression will cause the utility to fail completion, whereas reaching the SWITCHTIME limit will cause the utility to start its completion. DB2 can allow the utility to go into the last log iteration before the SWITCHTIME is met, if it can honour the original MAXRO value.

NEW DRAIN_ALLPARTS PARAMETER

Should this not be enough, DB2 now allows a new option on the reorgs statement that causes a drain on all partitions during the switch phase rather than partition by partition. Within the drain specifications, the option DRAIN_ALLPARTS controls how DB2 will process its drain request for a partitioned table space with non-partitioned index.

```
REORG DB.TS .... DRAIN_ALLPARTS NO
```

This is the default and is similar to the behaviour of the previous releases of DB2. This means that DB2 will drain the target partitions serially followed by the non-partitioned secondary indexes.

The problem that can arise with this way of working is that a DML (insert/update/delete) statement can follow the reverse physical order of the reorg causing drain timeouts or deadlocks. In other words if the reorg is first handling partition 1, then 2, then 3 but an update is starting on partition 3, then 2, then 1, the default behaviour will cause problems.

```
REORG DB.TS .... DRAIN_ALLPARTS YES
```

This will solve that issue as DB2 will first take a table space level drain on the entire partitioned table space. By doing this, DB2 prevents interfering SQL to start. DB2 11 introduces better control for partition parallelism during LISTDEF processing. Before this version partition parallelism for LISTDEF processing was an all or nothing choice, either parallelism was allowed or not. This could cause certain users not to use parallelism as they cannot afford to shadow many partitions at a time.

LISTPARTS OPTION

DB2 11 introduces a more granular way of controlling parallelism by way of the keyword LISTPART

```
REORG LIST …. LISTPARTS n
```

LISTPARTS specifies the maximum number of data partitions to be reorganized in a single

REORG on a LISTDEF that contains PARTLEVEL list items.

In DB2 11 PARALLEL YES/NO is still supported but is deprecated. In order to mimic PARALLEL NO, the reorg statement could read

```
REORG LIST …. LISTPARTS 1
```

In order to match PARALLEL YES the LISTPARTS keyword can simply be omitted. Any intermediate granularity is the user's choice. This option is primarily used to reduce the resource consumption (in particular DASD for shadows and sortwork datasets) by breaking up the REORGs into multiple executions. The best performing REORG, in terms of total CPU and Elapsed Time, is the REORG that processes all target data partitions in a single execution.

UNLOAD BEHAVIOR

Finally it's worth pointing out a change in behaviour for data being unloaded using the REORG utility. Before DB2 11 when a REORG DISCARD was used or when a REORG UNLOAD EXTERNAL was used that variable length data was padded to its maximum length. However starting in DB2 11 this will no longer be the case. Discarded data will now be non-padded by default:

```
REORG .... DISCARD NOPAD YES
```

and the unload external option will now as default behave like:

```
REORG .... UNLOAD EXTERNAL NOPAD YES
```

For those users that are still using the INDREFLIMIT and the OFFPOSLIMIT keywords instead of real time statistics (RTS), now might be the time to start looking at RTS as many of the early day's conditional parameters will we deprecated in DB2 11.

THE RUNSTATS UTILITY

Every new release of DB2 relies on more accurate statistics to make the best possible access path decisions. The runtime cost of gathering these more detailed statistics could, in some cases, be a relatively costly endeavour, especially if distribution statistics were needed. DB2 10 brought relief by making some RUNSTATS gathering ZIIP eligible. DB2 11 brings further relief as the gathering of distribution statistics as well as inline statistics become more ZIIP offload-able.

The inline statistics become more powerful in DB2 11 allowing more options and reducing the need to run full statistics. One inline statistics enhancement is the possibility to run inline statistics on non-partitioned indexes during a REORG utility, to collect inline statistics for an entire NPI, on a partition level REORG the keyword SORTNPSI must be specified. SORTNPSI YES|AUTO (or its ZPARM equivalent) is a pre-requisite for a partition level Online REORG to gather statistics on a NPI. However, it is not an if-and-only-if relationship. REORG can choose to not sort all the NPI keys when SORTNPSI YES|AUTO is specified based on certain internal threshold, and a part level Online REORG cannot gather NPI statistics inline unless all the NPI keys are sorted.

Other inline statistic improvements (during LOAD, REORG and REBUILD INDEX) are full support for distribution statistics. It is now possible to collect COLGROUP and HISTOGRAM information.

The most important improvement to the RUNSTATS utility however is the ability of the DB2 11 optimizer to indicate which statistics it is missing. This could then be used as input to running statistics and determining the statistics profile that should be used for an object.

The way this works when owning all needed tooling:
The optimizer identifies missing statistics and writes this information out to new catalog tables. Optim Query Workload Tuner modifies the statistics profile for those objects. The automation tool detects a change of profile and will rebuild its statistics jobs.

Should the needed tools not be available, the DSNACCOX stored procedure has been improve to recommend the needed RUNSTATS, allowing users to write and update their own RUNSTATS mechanism.

If no profile exists for a table exists in DB2 11, the following default statistics will be collected at RUNSTATS time. When a table name is not specified, TABLE ALL INDEX ALL is used for the profile specification. When a table name is specified, COLUMN ALL INDEX ALL is used for the profile specification.

A completely new feature is the ability to reset the existing statistics during a RUNSTATS utility, rather than manually trying to delete those statistics when they are no longer needed or accurate. The syntax to execute this:

```
RUNSTATS TABLESPACE ... RESET ACCESSPATH
```

This runstats option resets access path statistics for all tables in the specified table space and related indexes. Real time statistics and space statistics in the catalog for the target objects are not reset. Be careful, recovery of previous values is not possible after the RUNSTATS utility is invoked with the RESET ACCESSPATH option.

RESEST ACCESSPATH does not reset the statistics currently in the _HIST tables for that object. Maintaining the _HIST tables during regular RUNSTATS invocation, is a good practice for those users wanting to use the RESET ACCESSPATH option.

Using the option HISTORY ACCESSPATH, provides the possibility to write out to the _HIST tables, when the RESET was done. It doesn't write out the values that were deleted, that would have to be done when collecting the values. It does not provide a method of recovering previous values. The utility option to do this is:

```
RUNSTATS TABLESPACE .... RESET ACCESSPATH HISTORY ACCESSPATH
```

It is evident that statements that refer to the objects for which statistics are reset are invalidated in the dynamic statement cache.

THE LOAD UTILITY

In DB2 11 it becomes possible to load multiple partitions in parallel using a single SYSREC input file. This avoids having to make the choice between processing the SYSREC dataset and splitting it up in multiple files on one hand, and abandoning parallelism and having a longer elapsed run time on the other hand. This improvement is available both for online (SHRLEVEL CHANGE) as well as offline (SHRLEVEL NONE) loads.

```
LOAD DATA .... PARALLEL n ...
```

For a single input data set, LOAD SHRLEVEL NONE PARALLEL has the capability to drive parallelism to process the single SYSREC input data in parallel, but it does not actually load the data records into multiple partitions concurrently in the back end.

Graphically a SHRLEVEL NONE load with PARALLEL 5, with a single SYSREC. The parallelism refers to the conversion tasks.

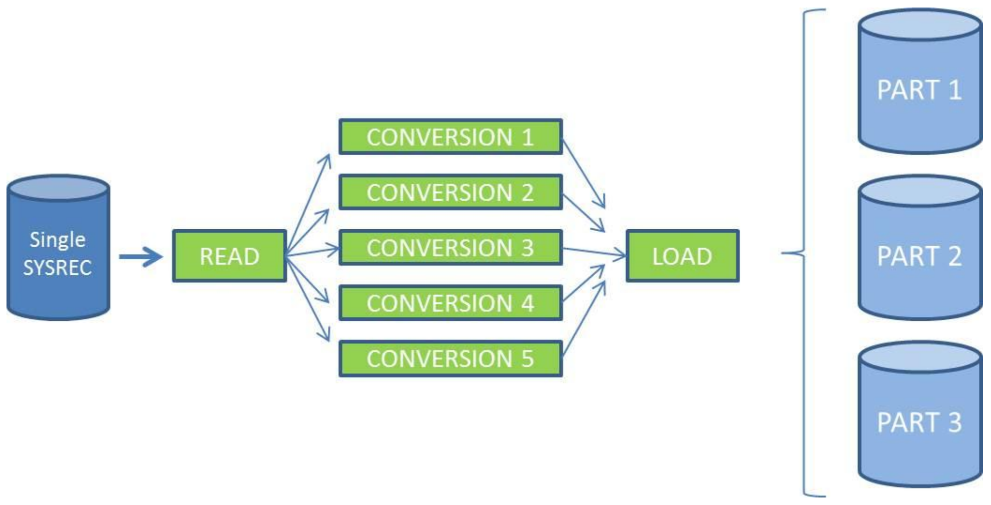


Figure 17 – LOAD DATA SHRLEVEL NONE with PARALLEL 5

By providing the keyword RBALRSN_CONVERSION, the load supports the conversion between basic and extended RBA LRSN format. It specifies the RBA or LRSN format of the target object after the completion of the LOAD utility. The possibilities are “none”, meaning no conversion is done, “basic” meaning if the target object is 10 bytes RBA LRSN it will be converted to 6 bytes and “extended” meaning if the target object is 6 bytes RBA LRSN it will be converted to 10 bytes If the keyword is not specified, the conversion specified in the UTILITY_OBJECT_CONVERSION subsystem parameter is accepted.

Users that quite often use the very practical cross loader functionality will be happy to learn that it now supports XML columns. In case of cross loading XML or a LOB column, DB2 11 now exploits Fetch Continue for cross loader for XML and LOB columns. This reduces the virtual storage requirements.

As mentioned in the Runstats utility improvements, the inline statistics options have been expanded to provide better distribution and histogram statistics, reducing the need to run supplementary runstats after the load utility.

THE REPAIR UTILITY

DB2 11 provides a new functionality to check for and solve (where possible) inconsistencies between the information in the catalog and the data. Before DB2 11 it could be very difficult to find out the differences between the object definition in the catalog and what really existed in the object (think page size, segsize, version number etc).

```
REPAIR CATALOG TABLESPACE dbname.tsname
```

This repair statement compares the row format, RBA format, data version and hash space value in the catalog with the data. If any differences for these values are found, the repair utility will the change the values in the catalog to match the data.

Beyond that, the REPAIR utility also validates DBID, PSID, and OBID, as well as table space type, SEGSIZE, PAGESIZE and table definitions. If any of these differences are found then REPAIR does not correct the information in the catalog.

Instead, REPAIR fails and reports the mismatched information in a message. To correct the mismatched information, take the action that is documented for the message that you receive.

```
REPAIR CATALOG TABLESPACE dbname.tsname TEST
```

The keyword TEST is optional in the utility statement by providing it, the repair utility will now only report the found differences but will not attempt to solve the differences, if any are found.

UTILITY RELATED COMMANDS

Although technically not utility improvements, some DB2 commands have been altered or added to provide us with more and better information. One enhancement is to include the jobname in the display utility command. Although not world shocking, it can be immensely practical.

Similarly practical is a command to externalize the real time statistics, rather than stopping and starting an object, or forcing a system checkpoint. The command to this is:

```
ACCESS DATABASE .... MODE(STATS)
```

DB2 11 brings some practical and nice to have improvements as well as highly important availability and performance solutions. This section gave an overview of some of the new possibilities of the DB2 11 utility suite. Let it be a start to fully exploiting DB2 11 once installed.

SECURITY ENHANCEMENTS

CREATE MASK ENHANCEMENTS

The CREATE MASK statement creates a column mask at the current server.
A column mask is used for column access control and specifies the value that should be returned for a specified column. One of the more known uses is for displaying only a part of a credit card number or SSN or displaying *** instead of salary, etc.
The mask can be used together with a security function to allow access by group.

Example:

```
CREATE MASK SSN_MASK ON EMPLOYEE
FOR COLUMN SSN
RETURN
CASE
WHEN (VERIFY_GROUP_FOR_USER(SSESSION_USER,'PAYROLL') = 1)
THEN SSN
WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'MGR') = 1)
THEN 'XXX-XX-' || SUBSTR(SSN,8,4)
ELSE NULL
END
ENABLE;
```

Starting with DB2 11 we have two more table types covered by mask:

- An archive-enabled table
- An archive table

The RETURN expression in the MASK can now handle also

- A table function
- A collection-derived table (UNNEST)

Please note that there are some new restrictions involving pending definition changes:

CREATE MASK is not allowed if the mask is defined on a table or references a table that has pending definition changes.

CREATE PERMISSION

The CREATE PERMISSION statement creates a row permission for row access control at the current server. It is used to control access to a full row within the database.

Example:

```
CREATE PERMISSION SALARY_ROW_ACCESS ON EMPLOYEE
FOR ROWS WHERE
      VERIFY_GROUP_FOR_USER(SESSION_USER,’MGR’,’ACCOUNTING’) = 1  AND
      ACCOUNTING_UDF(SALARY) < 120000
ENFORCED FOR ALL ACCESS
ENABLE;
```

Starting with DB2 11 we have two more table types covered by Row Permission:

- An archive-enabled table
- An archive table

The FOR ROWS WHERE expression in the PERMISSION can now handle also

- A table function
- A collection-derived table (UNNEST)

(*) There are no changes (from DB2 10) to Create Role / Create Trusted Context.

RACF-MANAGED SECURITY ENHANCEMENTS

DB2 10 & RACF EXIT

The current (DB2 10) RACF exit in DB2 (DSNX@XAC) has some major shortcomings

The OWNER keyword is not honoured when RACF exit is used to control authorization. RACF checks the invoker (primary-authorization ID) and not the owner. This applies to BIND, REBIND and AUTOBIND (if allowed in ZPARM).
Moreover, if we use ABIND=YES in ZPARM and the invoker is not authorized to bind the program then we get an application failure and need DBA intervention. We can choose to use ABIND=NO but still remain with program failure and need of manual intervention.

Another problem is the usage of DYNAMICRULES(BIND) option with Dynamic SQL – this does not work as well as RACF checks the invoker ID regardless of DYNAMICRULES specification and RACF authorization fails if the invoker is not authorized to execute the SQL statements in the package.

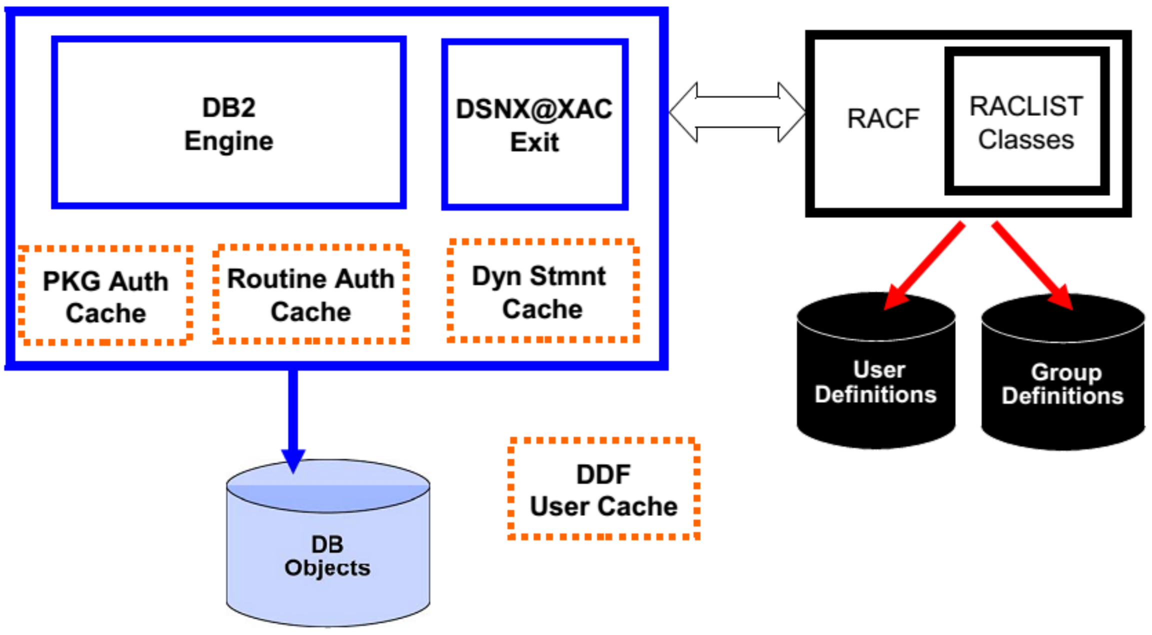


Figure 19 - DB210 and RACF Access Control Authorization Exit (ACAE) Authorization

Another issue is that Dynamic statement cache and authorization caches are never flushed at RACF authority change. This means that DB2 cache may retain privileges after RACF REVOKE and the DBAs must take manual action (GRANT/REVOKE/RUNSTATS) in order to sync with RACF. The DDF authentication cache is automatically refreshed every 3 minutes, but this still creates a potential short-term issue following any RACF changes.

DB2 10: Sync DB2 Cache on RACF GRANT/REVOKE

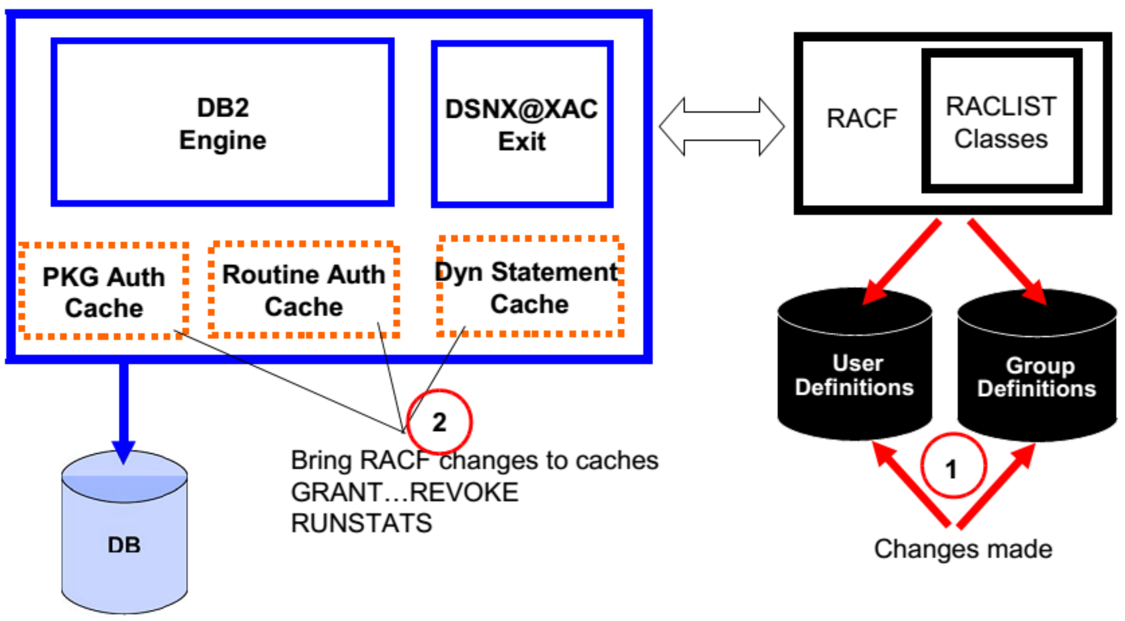


Figure 20 – DB2 10 RACF/DB2 Co-Ordination

DB2 11 PACKAGE AUTHORIZATION ENHANCEMENTS - REVOLUTION

In DB2 11 AUTOBIND, BIND, REBIND all provide PKG-owner ACEE to RACF thus allowing usage of the OWNER keyword. Its' value may be RACF ID, GROUP or ROLE and RACF will ACCEPT the request.

Dynamic SQL authorization checking supports the authorization-ID as specified by the DYNAMICRULES value, be it the package owner, or the user-ID that either executes the package or defines the package or invokes the package. This gives us a simple and versatile support of security and role separation.

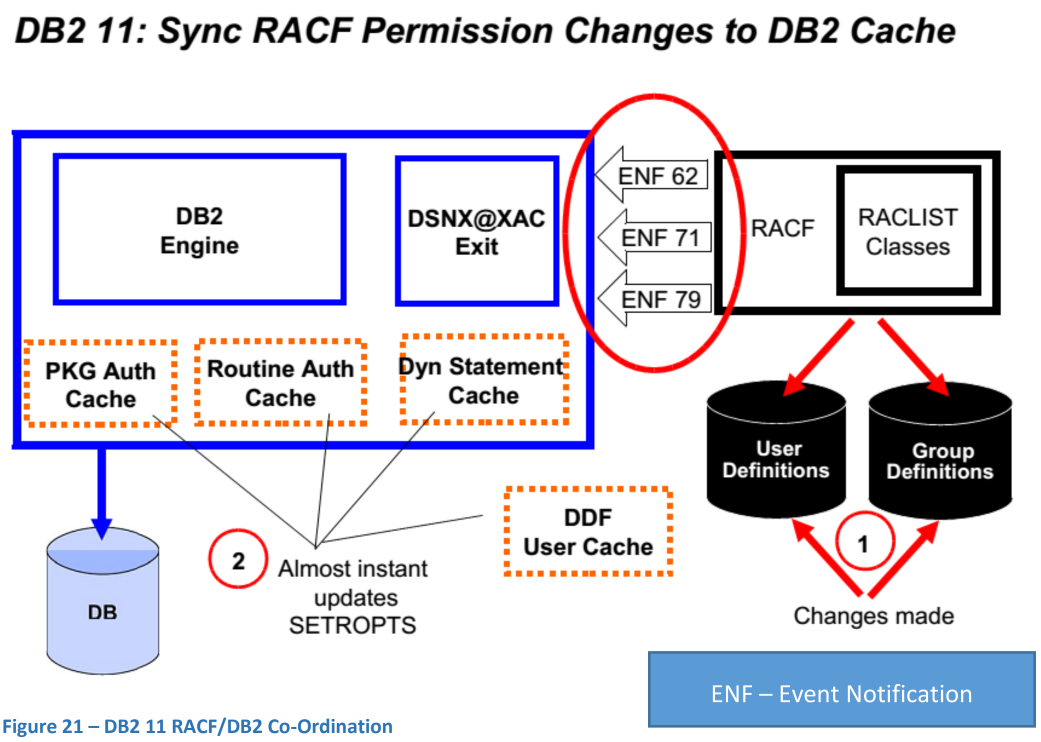


Figure 21 – DB2 11 RACF/DB2 Co-Ordination

The bottom line is that now we have similar behaviour between DB2 authorization and RACF authorization.

DB2 11 INSTALLATION PARAMETER – AUTHEXIT_CHECK

With DB2 11 comes a new ZPARM - AUTHEXIT_CHECK - which governs DB2 interaction with RACF:
It can be either DB2 or PRIMARY and it allows us to specify whether the owner or the primary authorization ID is to be used for authorization checks.
Using DB2 (DB2 11 behaviour) provides the ACEE of the owner, to RACF for AUTOBIND/BIND/REBIND as well as providing the ACEE of the authorization ID as specified by the DYNAMICRULES value.

Using PRIMARY (default – DB2 10 behaviour) provides the ACEE of the primary authorization ID for all authorization checks.

This parameter cannot be changed online ☹

Pay attention to the fact AUTHEXIT_CHECK=DB2 uses the XAPLUCHK value for authority checks and if we use a group or role in RACF we need to give it the proper permission (same applies to dynamic SQL with DYNAMICRULES value other than RUN):

E.g. PERMIT DSN.BATCH CLASS(DSNR) ID(DB2GROUP) ACCESS(READ)

RACF ENF SIGNALS HEARD BY DB2 11

RACF generates Event Notifications (ENF), when a profile is changed. DB2 11 listens for the following ENF:

- ENF 62: RACF options refreshed
  - SETROPTS RACLIST REFRESH
- ENF 71: User permissions changed; DB2 refreshes cache entry
  - ALTUSER REVOKE, CONNECT REVOKE, DELUSER, DELGROUP, REMOVE

- ENF 79: User permissions to access resource changed
  - PERMIT..DELETE, ACCESS(NONE), RESET, WHEN(CRITERIA(SQLROLE...))
  - RALTER.. UACC(NONE), DELMEM; RDELETE

When DB2 11 receives ENF 79, it stores the changes and waits for ENF 62 to actually refresh the cache entries.

RACF requirements are to have the RACF class descriptor table with the value of SIGNAL=YES in order to activate the ENF 62 and 79. We also need z/OS 1.13 and APARs OA39486, OA39487, OA39506.

Those who use other security products (TopSecret / ACF2) should check with their ISV whether those products conform to ENF generation.

DB2 11 INSTALLATION PARAMETER – AUTHEXIT_CACHEREFRESH

There is a new parameter to govern DB2 interaction with RACF which allows to specify whether the caches are to be refreshed when user profile or resource access is changed in RACF.

AUTHEXIT_CACHEREFRESH (ALL or NONE)

- ALL (DB2 11 behaviour)
  - DB2 listens for ENF 62, ENF 71 and ENF 79 signals
  - Package authorization cache, Routine authorization cache and dynamic statement cache entries are refreshed
- NONE(default)
  - The cache entries are not refreshed.

Unfortunately, there is no online update of this parameter.

You need to remember that profile names with generic character * or ** and has less number of parts than supported by the CLASS parameter other than DSNADM class thus all objects and/or all privileges for the specified CLASS parameter may be considered for cache refresh. In the DSNADM class, using ID(Authid) deletes all the entries in the caches for the specified Authid.

DB2 11: OTHER RACF SUPPORT

DSNXRXAC RACF access control also supports global variable privileges like READ(READAUTH) and WRITE(WRITEAUTH). AUTOBIND for UDFs can also benefit from having all RACF checks thus we shall not get RC8, with reason code 17 on AUTOBIND for UDFs.

DB2 11: NEW RACF INSTRUMENTATION ENHANCEMENTS

- IFCID 106 now supports tracing of ZPARM parameters:
  - AUTHEXIT_CHECK
  - AUTHEXIT_CACHEREFRESH
- New IFCID 374 serviceability trace cut on:
  - Entry added to/deleted from package or routine authorization cache
  - Search of package or routine authorization cache
- New IFCID 386 serviceability trace is written on receipt of RACF ENF signal

MESSAGES AND CODES

- SQLCODE -551 updated:
  - May occur if ACEE cannot be created for AUTHID
- Each of these messages may occur when AUTHEXIT_CHECK=DB2 and ACEE cannot be created for the authorization ID:

- DSNT210I: AUTHID lacks privileges, cannot BIND
- DSNT235I: Authorization error using specific privilege
- DSNT241I: Authorization error on package execution
- DSNX101I: AUTHID lacks privilege for plan execution

- New DSNX235E - Registration of the ENF listener exit failed
    - Specifies the signal number: 62, 71 or 79.

INSTRUMENTATION ENHANCEMENTS

With the convergence of the temporary work files (work files used by SORTs, work files needed for cursor processing, and Declared Global Temporary tables), customers are faced with the tasks of sizing their work file database, and setting the value for MAXTEMPs. DB2 11 provides significant relief in this area, with the addition of several new fields, and additional warning messages. Two new ZPARMs WFSTGUSE_AGENT_THRESHOLD, and WFSTGUSE_SYSTEM_THRESHOLD provide alerts using DSNI052I and DSNI053I messages indicating that a pre-determined amount of the workfile is under use. Several additional fields (QISTAMXU) have been added to IFCID 0002 to track work file utilization.

In large data sharing environments, with high concurrent batch activity, sizing of Group Bufferpools is critical. A smaller buffer pool or improper setting of CLASST can result in group Bufferpool getting critical which in turn will lead to a system slow down. The write-around protocol enables DB2 to write directly from the Group Bufferpool to disk essentially eliminating a hot-spot. Additional fields have been added to IFCID 0002 (QBGLWA) and IFCID 0003 (QBGAWA) to track the number of changed pages that were written to DASD using the group Bufferpool write-around protocol.

Feature Description (technical description of new feature/function)

DB2 11 INSTRUMENTATION CHANGES

DB2 11 for z/OS introduces several trace enhancements.

The following table provides an overview of the instrumentation changes in DB2 11.

Enhancement	Description of changes
Array support	Fields are added to record information about storage use by arrays
Autonomous transactions	Fields are added to record information about autonomous transactions, and on their effect on parallel groups
Application compatibility	Field values are added to record incompatibilities between Version 11 and previous DB2 versions
Castout enhancements	Fields are added to record class castout queue threshold values, based on the number of pages
Larger RBA and LRSN support	Fields that contain RBAs and LRSNs have been expanded from six or eight bytes to 10 bytes
Parallelism performance enhancements	Fields are added to track the effect of changes to the degree of parallelism after parallel system negotiation that occurs because of resource constraints
Temporal support	Fields are added to indicate the impacts of the CURRENT TEMPORAL BUSINESS_TIME special register, the CURRENT TEMPORAL SYSTEM_TIME special register, and the SYSIBMADM.GET_ARCHIVE built-in global variable

These new IFCIDs are shipped with DB2 11:

- 0366: Records use of incompatible SQL and/or XML for this release of DB2. Please refer to Impact Analysis for Incompatible Changes on page 34 for more details.
- 0376: A roll up of 366, with one record written for each unique static or dynamic statement
- 0377: Records information about indexes for which pseudo-deleted index entries are automatically cleaned up
- 0382: Records suspend operations for parallel task synchronization
- 0383: Records resume operations after parallel task synchronization

These IFCIDs have been modified in DB2 11

- 0002, 0003, 0316, 0401: Fields have been added to track the effect of changes to the degree of
- Parallelism after parallel system negotiation that occurs because of resource constraints.
- 0002: Additional fields have been added to track work file usage information at an agent level and a system level.
- 0002, 0003: Additional fields have been added to track the number of pages written to disk using group-buffer-pool write-around protocol.

This example shows a part of an OMEGAMON XE for DB2 Performance Expert on z/OS Accounting Trace Long report. This report illustrates the instrumentation support of the client info longer fields. In these reports, note the ENDUSER, TRANSACT and WSNAME fields.

```
---- IDENTIFICATION --------------------------------------------------------------------------------------------------------
ACCT TSTAMP: 08/01/13 22:36:37.38    PLANNAME: DB211Jav        WLM SCL: DDFBAT          CICS NET: N/A
BEGIN TIME : 08/01/13 22:36:15.07    PROD TYP: JDBC DRIVER                              CICS LUN: N/A
END TIME   : 08/01/13 22:36:37.38    PROD VER: V4 R17M0        LUW NET: G9378921        CICS INS: N/A
REQUESTER  : ::9.55.137.33           CORRNAME: db2jcc_a        LUW LUN: D8D9
MAINPACK   : DB211Jav                CORRNMBR: ppli            LUW INS: CBBFAC100F80    ENDUSER : ClientUser_012#1
PRIMAUTH   : DB2R1                   CONNTYPE: DRDA            LUW SEQ:            2    TRANSACT: DB211JavaNewDriver_01234567890#1
ORIGAUTH   : DB2R1                   CONNECT : SERVER                                   WSNAME   : WorkstationName_#1
```

The following section shows how the truncated values are expanded in the report to show the complete values.

```
    ----------------------------------------------------------------------------------------------------------------------------------
    |TRUNCATED VALUE                  FULL VALUE                                                                                     |
    |ClientUser_012#1                 ClientUser_0123456789012345678901234567890123456789012345678901234567890123456789012345678901|
    |                                 23456789012345678901234567890123456                                                          |
    |DB211JavaNewDriver_01234567890#1 DB211JavaNewDriver_01234567890123456789012345678901234567890123456789012345678901234567890123|
    |                                 4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456|
    |                                 7890123456789012345678901234567890123456789                                                  |
    |WorkstationName_#1               WorkstationName_01234567890123456789012345678901234567890123456789012345678901234567890123456|
    |                                 789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789|
    |                                 0123456789012345678901234567890123456789                                                     |
    ----------------------------------------------------------------------------------------------------------------------------------
```

SEE ALSO

Please see below for a cross reference to other topics that may also be of interest to database administrators, but have been placed elsewhere within this document.

- Extended log record addressing on page 7
- DRDA/DDF Enhancements on page 7
- Buffer Pool enhancements on page 12

- Data Sharing Enhancements on page 12
- SELECT from Directory Pagesets on page 13
- Additional zIIP Enablement on page 14
- Installation on page 16
- Migration on page 17
- Transparent Archive Query on page 56
- SQL Enhancements on page 61
- Analytics Enhancements on page 69
- Global Variables on page 78
- Variable arrays on page 80
- Declared Global Temporary Table Enhancements on page 83

DEVELOPER TOPICS

This section of the document discusses new DB2 11 features that are expected to be of most interest to DB2 Developers.

TRANSPARENT ARCHIVE QUERY

INTRODUCTION

Throughout the history of DB2 on z/OS, DB2 development has continuously improved the product so that larger quantities of data can be maintained in a single table. Nonetheless, there is certainly value in separating data that is still subject to change or referenced very frequently from historical (archive) data that is no longer subject to change and accessed very rarely.

Separating operational from the archive data in separate tables provides increased efficiency for applications. The operational table and indexes for the application process is to a much smaller extent burdened by non-operational data. This also applies to housekeeping processes. In addition, housekeeping processes can run at a much lower frequency for non-operational data, creating another opportunity to reduce data management cost and elapsed time.

Before DB2 11, it was up to the user to organize not only the archiving process, but also ensure proper coding of SQL to include or exclude the archive data. Splitting an existing table into an operational table and an archive table thus meant reviewing and recoding of all SQL statements against the original table.

In many cases, the requirement to optimize access to the operational data is recognized well after the initial creation of the applications. The user community may have developed a wealth of SQL queries against the table. As such, turning a table into a combination of an operational table and an archive table will be associated with a lot of effort.

The situation could be depicted as follows:

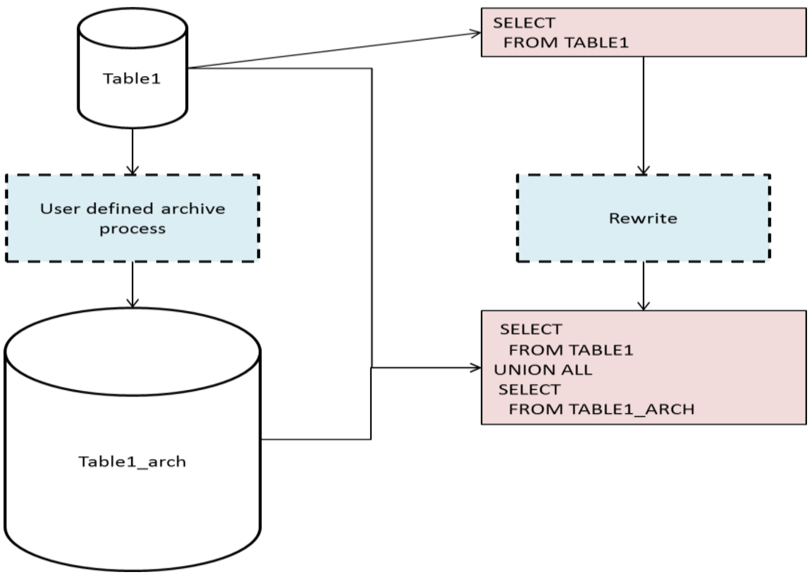


Figure 22 - Manual Archiving Example

The user defined archiving process would have to read the data to be archived from the operational table (Table1), insert into the archive table (Table1_arch) and delete from the operational table.

Since DB2 is completely oblivious of the connection between the base table and the archive table, the end user is also responsible for coding the SQL correctly to access the operational table, the archive table or a combination of the operational and archive tables.

Further, the user is also responsible for ensuring the layout of the operational table and the archive are kept in sync. Changes must be applied to both tables as well as the archiving process.

If the identification of data to be archived is simple (i.e. can be expressed with stage 1 predicates) then the cheapest way to implement the user defined archive process is usually using a combination of REORG DISCARD of the operational tablespace and a subsequent LOAD of the discarded rows into the archive. An additional advantage of a REORG/LOAD based process is that the base tablespace and its statistics are in pristine condition afterwards (presuming, inline image copy and inline stats are included in the REORG).

SO WHAT IS TRANSPARENT ARCHIVING?

DB2 11 Transparent Archiving is built on the infrastructure that was introduced in DB2 V10 for (bi) temporal support. Transparent archiving allows you to define a table, the archive table, and associate that with an existing table. The latter table is then referred to as the archive enabled table.

Since DB2 is now aware of the connection between the operational table and the archive table, DB2 will take care of most of the activities associated with archiving as can be seen below.

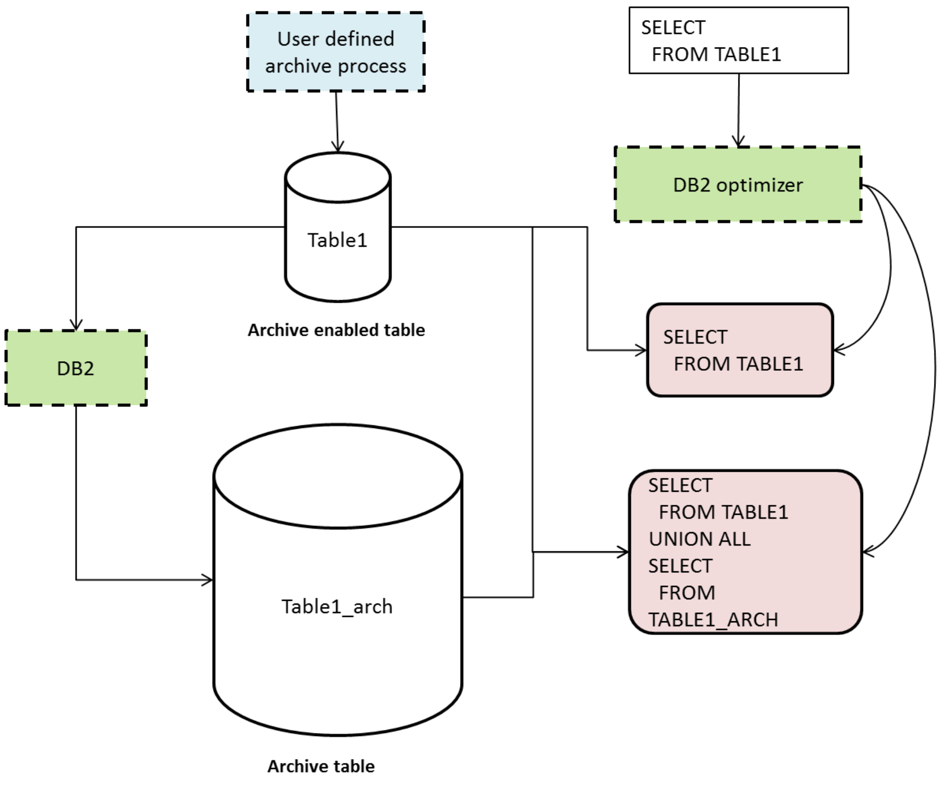


Figure 23 - Transparent Archiving Example

The user is still responsible for defining the archiving process. However, the process now only needs to determine what data to archive and simply issue a 'DELETE' for that data. The transport from the operational table to the archive is taken care of by DB2.

When an SQL statement deletes a row from the archive enabled table, DB2 will automatically insert it into the archive table. There is a new system defined global variable SYSIBMADM.MOVE_TO_ARCHIVE (with default = 'N') that allows the user to enable or disable this behaviour so that truly unwanted rows can still be deleted from the operational table without automatically appearing in the archive.

The process above will also work if the data is deleted from the archive enabled table as the result of a cascading delete of a parent table. Of course, a DB2 enforced referential constraint must be defined.

The archiving method using REORG DISCARD and LOAD is also available. Unfortunately, REORG DISCARD does not have an option to transfer the discarded rows directly to the archive table. This would actually make a lot of sense and let's hope DB2 development will provide this in the future.

Even more exciting is the fact that the same SQL statement can be used to retrieve data from the operational table or from a combination of the operational table and the archive. The latter option ensures that the result set of SQL statements can remain the same before and after the enabling of an archive solution. The impact on the end user is thus massively reduced.

When selecting from the archive enabled table, the user can control whether or not the data contained in the archive table is automatically included. If the system defined global variable SYSIBMADM.GET_ARCHIVE (with default = 'N') has been set to 'N', only the table specified in the SQL statement will be examined. If this variable is set to 'Y', DB2 will automatically transform the SQL statement to use a union of the archive enabled table and the archive table.

There is no performance impact caused by Union All if application requires only the operational data. This is achieved by optimizing the same static SQL statement twice during bind, one regular section without Union All expansion, and one extended section with the Union All expansion. At execution time, if SYSIBMADM.GET_ARCHIVE = 'N', the regular section is executed, and if SYSIBMADM.GET_ARCHIVE = 'Y', the extended section is executed. In other words, the added flexibility of archive transparency will not impact the SQL performance when accessing operational data only. For dynamic SQL, the SQL is optimized based on the value of SYSIBMADM.GET_ARCHIVE which determines whether to perform Union All expansion.

Note: The package must have been bound with ARCHIVESENSITIVE(YES) in order for DB2 to take the value of SYSIBMADM.GET_ARCHIVE into consideration. For SQL scalar function or SQL native stored procedure, the option must be ARCHIVE SENSITIVE YES.

As a result, DB2 is able to shield the user from having to recode SQL statements to distinguish between queries that access the operational data only or a combination of the operational data and the archived data.

DB2 will also support addition of new columns but more complicated changes are not supported while the archiving relationship exists. In order to implement such changes, the archiving relation must be disabled first.

ENABLING TRANSPARENT ARCHIVING

Let's follow steps that need to be taken to split an existing table into an archive enabled table (the original) and an archive table.

The existing table looks like this:

```
CREATE TABLE
  T412.TR400002
   (
      PGMID  CHAR(8)  NOT NULL
      ,VERARB  CHAR(1) NOT NULL
      ,BUCHEN  CHAR(1) NOT NULL
      ,KKONL  CHAR(1)  NOT NULL
    )
```

The first step to take is to define an archive table that will be associated with (what will become) the archive enabled table. Important is that the columns are identical. Should you use "CREATE TABLE LIKE" to achieve this, be aware that identity columns, row change timestamp columns etc. need special clauses on the CREATE statement to ensure the attributes of such columns are the same in the archive table (see the 'copy options' of the CREATE TABLE LIKE syntax in the SQL reference).

The next step is to associate the archive table with the existing table. This is done by executing:

```
ALTER TABLE T412.TR400002
 ENABLE ARCHIVE USE T412.TR400002_ARCH
```

At this point we have an archive enabled table and an associated archive table that DB2 is aware of. This can be verified by examining SYSIBM.SYSTABLES (ARCH_SCHEMA, ARCH_TABLE)

```
SELECT SUBSTR(CREATOR,1,8)              AS CREATOR
      ,SUBSTR(NAME,1,15)                AS NAME
      ,SUBSTR(ARCHIVING_SCHEMA,1,8)     AS ARCH_SCHEMA
      ,SUBSTR(ARCHIVING_TABLE,1,15)     AS ARCH_TABLE
  FROM SYSIBM.SYSTABLES
 WHERE CREATOR = 'T412'
   AND NAME LIKE 'TR40%'
```

CREATOR	**NAME**	**ARCH_SCHEMA**	**ARCH_TABLE**
T412	TR400002	T412	TR400002_ARCH
T412	TR400002_ARCH	T412	TR400002

USING THE TABLES

To illustrate how archiving and selecting works, I inserted three rows in the archive enabled table. The T412.TR400002_ARCH table is empty. The contents of the archive enabled table are now as follows:

```
SELECT * FROM T412.TR400002;

PGMID    VERARB  BUCHEN  KKONL
TEST04   2       2       2
TEST05   2       2       2
TEST06   2       2       2
```

Subsequently the following two SQL statements were executed:

```
SET SYSIBMADM.MOVE_TO_ARCHIVE = 'Y';

DELETE FROM T412.TR400002
  WHERE PGMID = 'TEST04';
```

Now the tables look like this:

```
SELECT * FROM T412.TR400002;
PGMID     VERARB  BUCHEN  KKONL
TEST05    2       2       2
TEST06    2       2       2
```

```
SELECT * FROM T412.TR400002_ARCH;
PGMID     VERARB  BUCHEN  KKONL
TEST04    2       2       2
```

After ensuring the package (SPUFI) was bound with ARCHIVESENSITIVE(YES), and instructing DB2 to include the archive data using the global system variable, the result set is as follows.

```
SET SYSIBMADM.GET_ARCHIVE = 'Y';
SELECT * FROM T412.TR400002;

PGMID     VERARB  BUCHEN  KKONL
TEST05    2       2       2
TEST06    2       2       2
TEST04    2       2       2
```

FINAL NOTES

DB2 has brought many improvements in the area of online schema development. Situations still remain where a drop is required. It is important to realize that if an archive enabled table is dropped, the archive table is dropped nicely along with it. If this is not your intent, please ensure to disable the archiving relationship.

The implementation requires an application to both bind the package with ARCHIVESENSITIVE (YES) and set the global variable SYSIBMADM.GET_ARCHIVE to 'Y'. One could argue that one could be enough. Let's assume the bind parameter would not exist. The effect would be that if a module somewhere in the calling path would set the global variable SYSIBMADM.GET_ARCHIVE to 'Y', then all of a sudden all packages in the thread would be affected. It is after all a global variable. So, without being touched at all, the module would change its behaviour. With the ARCHIVESENSITIVE bind parameter, we get additional control to avoid that situation.

Similarly, one could question why setting another global variable (SYSIBMADM.MOVE_TO_ARCHIVE) is necessary in applications that delete from an archive enabled table. To be fair, this opens the opportunity that things go wrong and the archive is no longer complete. Then again, if moving data would not be under control of a global variable, deleting truly unwanted data could become difficult. In order to remove the unwanted data from the archive enabled table (that should not end up in the archive), one would have to disable the archiving and perform the delete. But what if you have more applications that delete and expect the data to be moved to the archive? It must now be ensured that none of such application runs while the relation between the archive enabled table and the archive is disabled.

One thing that certainly must be considered is that fact if a row is now moved to an archive table, the primary key of that row becomes available again in the archive enabled table. It could occur that the same primary key (or any other unique key for that matter) is reinserted. This may or may not lead to problems.

It would be a welcome addition if IDAA would also support the transparent archiving solution in the future. Even though IDAA provides HPSS, this requires a strict partitioning scheme. Implementing such a scheme may not always be possible. Once IDAA is able to support SQL where one query block is executed by the DB2 engine while another query block is processed by the Netezza box, IDAA would become a logical target for the archive tables.

SQL ENHANCEMENTS

IMPROVED PERFORMANCE OF DUPLICATE REMOVAL

Query sort performance is always a concern, but many times DB2 users aren’t sure just what the performance impact of a sort can be. EXPLAIN can be used to indicate whether or not a sort is taking place, but in some situations the sort information is buried deep within the EXPLAIN tables. The only way to truly gauge the impact of a sort is to run benchmark tests of queries that may or may not avoid sorting in order to measure and estimate the true resource impact of DB2 sorts. Relatively few customers run benchmarks of individual queries in order to measure the impact of sort performance, and so many times sort costs are simply assumed or ignored. Over the course of several releases of DB2 IBM has worked towards reducing or eliminating the need for sorts in support of certain SQL constructs such as GROUP BY, DISTINCT, and certain types of subqueries. This reduces the need for users to attempt to vary SQL statements just to try to avoid sorts. In addition, for users that do not spend the time to understand the impact of sorts and attempt tuning, the improved performance of these types of queries have proved to be “free” cost savings. However, given the complexity of the DB2 engine, and the variety of SQL constructs support there remain opportunities to improve the performance of DB2 queries by either eliminating or reducing sort processing.

DB2 11 addresses the continued requirement for improved performance by introducing improved sort avoidance for certain types of query constructs:

- SELECT DISTINCT
- GROUP BY
- Single MAX or MIN aggregate function
- GROUP BY with a single MAX or MIN function
- Certain non-correlated subqueries (not transformed)

The way this is going to work for DISTINCT is that prior to DB2 11 the duplicate values from non-unique indexes (if a non-unique index was used and provides order) were read by the index manager and passed to sort where sort did not actually sort, but instead eliminated the duplicate values. Now, with DB2 11 those duplicate values will be removed by the index manager (again, if a non-unique index that provides order is used) and fewer values passed on in the query processing. Similarly for GROUP BY prior to DB2 11, it was the DB2 runtime component that removed the duplicate entries if the index was scanned in order and sort was not required. In DB2 11, index manager will skip over the duplicates and avoid passing them to later stages of query processing. These cases can change the cost analysis within the optimizer, which can influence things such as index selection and table access sequence. So, access paths can change as a result of this enhancement.

How do you know when this is happening? Well, it’s not so obvious since for most situations it will not show up in a PLAN_TABLE (it may for non-correlated subqueries). The evidence is actually in the explain table called DSN_DETCOST_TABLE. In versions prior to DB2 11 the sort was not exposed in the PLAN_TABLE, but for

DISTINCT, there was a row in the DSN_DETCOST_TABLE representing the cost of sort removing the duplicate values. For GROUP BY with sort avoided, there wasn't a row in any explain table, since the sort was avoided. With DB2 11 that extra cost (and row in DSN_DETCOST_TABLE) is removed from the EXPLAIN output. In its place will be a value set in a new column of DSN_DETCOST_TABLE called IXSCAN_SKIP_DUPS with a value of "Y" indicating that this new processing will occur.

A simple test of the following query against the DB2 sample table EMP proves the change if the non-unique index on the WORKDEPT column is used for the access path:

```
SELECT DISTINCT WORKDEPT from EMP;
```

In the case of the query EXPLAINed under DB2 10 the DSN_DETCOST_TABLE explain table contains two rows, one row representing the cost associated with passing all the qualifying rows from the index into the sort process at run time. Notice the number of rows processed is 42, which is the number of entries in the index.

```
SCCOLS  SCROWS              SCRECSZ     SCPAGES
1       4.200000E+01        4           3.307086E-01
```

Under DB2 11 there is only one row in DSN_DETCOST_TABLE reflecting the number of rows processed as 1 and indicating index skipping has occurred. All "SC" columns are set to zero as no runtime sort processing will be invoked.

```
IXSCAN_SKIP_DUPS
Y
```

As mentioned before this change in cost will have an effect on access path selection, and of course it will result in a reduction in CPU consumption for impacted types of SQL constructs.

QUERY TRANSFORMATION OF STAGE 2 PREDICATES

The SQL language is extremely powerful and flexible and really a significant programming language in its own right. However, when coding SQL statements, programmers are often focused on functionality and speed of delivery, and not necessarily on performance. So, when coding predicates in a query programmers are not typically thinking about how DB2 is going to process those predicates. They are interested, however, in whether or not DB2 will use an index in support of a query, and are sometimes surprised when it does not. There are two major components to the DB2 engine which are often referred to as stage 1 (sargable) and stage 2 (non-sargable or residual). The stage 1 component is primarily responsible for accessing indexes and tables, and filtering data from those indexes and tables. The stage 2 component does things such as joining data, sorting, aggregation, processing functions, and processing expressions. The stage 2 component doesn't process against indexes, but against the data it is delivered from the stage 1 component. This typically means that if a function or expression is coded in a predicate against a column of a table then the data from that column has to be read by the stage 1 component and then passed to the stage 2 component so that the function or expression can be evaluated and the predicate applied. As one can imagine this type of processing can preclude the use of indexes for filtering in some situations, thus leading to reduced performance for these types of queries. While the DB2 Performance Guide outlines the differences in indexable, stage 1, and stage 2 predicates, many programmers are not aware of these differences and the impact to processing as they are simply focused on solving a business problem. This can result in many cases in which stage 2 predicates have been coded, but could have been coded as stage 1 instead. By the time the potential rewrite is determined the application may have already been delivered and the likelihood of the query being rewritten by the developer reduced. If rewriting the query is not possible a DBA may elect to create an index on expression, but this is not always a practical choice. The final solution may be a more expensive process of changing the application, testing, and implementing the change. Some examples of stage 2 non-indexable predicates that can be rewritten as stage 1 indexable are as follows (with rewritten example):

- YEAR(<date column>) op value (this is stage 2 non-indexable)
  - <date column> op DATE(value) (this is stage 1 indexable)
  - <date column> BETWEEN value AND value (this is stage 1 indexable)
- DATE(<timestamp column>) op value (this is stage 2 non-indexable)
  - <timestamp column> op TIMESTAMP(value) (this is stage 1 indexable)
- Value BETWEEN < column> AND < column> (this is stage 2 non-indexable)
  - < column> >= value AND < column> <= value (this is stage 1 indexable)

DB2 11 has taken the responsibility of rewriting these types of stage 2 predicates out of the hands of developers by rewriting the predicates automatically during the query transformation portion of statement processing. Query transformation occurs during the first part of the SQL statement compilation processes when a dynamic statement is prepared or a static statement is bound. DB2 makes changes to the input query in an effort to improve the access path selection process. In the case of DB2 11 improvements have been made to the query transformation process such that some predicates that were stage 2, such as the examples above(YEAR(DATE_COL), DATE(TIMESTAMP_COL), value BETWEEN C1 AND C2, and also SUBSTR(C1,1,n)), will be automatically rewritten as stage 1 (and potentially indexable) predicates. This is yet another “free” performance enhancement delivered with DB2 11, and will result in improved index matching and a reduction in the number of indexes required.

If an index on expression exists for the original predicate, then the predicate will not be rewritten from stage 2 to stage 1. The target for this DB2 11 enhancement is stage 2 predicates that cannot currently exploit an available index, and the existence of an index on expression can already result in indexability for the predicate.

Whether or not a predicate is transformed from stage 2 to stage 1 is evident when an EXPLAIN is issued against a query containing one of these types of predicates. The transformed query is exposed in the EXPLAIN table called DSN_QUERY_TABLE, and is best viewed using the IBM Data Studio tool. In addition, the EXPLAIN table DSN_PREDICAT_TABLE contains predicate text including transformed predicates, and the EXPLAIN table DSN_FILTER_TABLE contains information as to which stage the predicate is processed including MATCHING, STAGE 1, and STAGE 2. For example, the following query contains what is traditionally a stage 2 predicate prior to DB2 11:

```
SELECT LASTNAME
FROM   DANL.EMP
WHERE  YEAR(BIRTHDATE) = 1954
```

DB2 is able to rewrite the query during query transformation into something like this:

```
SELECT LASTNAME
FROM   DANL.EMP
WHERE  BIRTHDATE = BETWEEN '1954-01-01' AND '1954-12-31'
```

The transformed predicate is now stage 1 indexable. If an index were to be created on the BIRTHDATE column that index would be considered for matching index access. In addition to IBM Data Studio displaying the transformed query, The DSN_PREDICATE_TABLE would contain the following for the predicate text:

```
TEXT
---+---------+---------+---------+---------+---------+--
EMP.BIRTHDATE BETWEEN '1954-01-01' AND '1954-12-31'
```

The DSN_FILTER_TABLE would show that the predicate is either stage 1 or matching. In our tests an index was created on the BIRTHDATE column so the STAGE column of DSN_FILTER_TABLE reflects the index usage for the predicate:

```
STAGE
------+-
MATCHING
```

IMPROVED PERFORMANCE OF SOME PREDICATES CONTAINING "IN" AND/OR "OR"

Compound predicates have historically introduced certain performance challenges, especially when it comes to non-Boolean term predicates. Traditionally, a Boolean term predicate is a simple or compound predicate that, when it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row. For example the following is a Boolean term compound predicate.

```
WHERE LASTNAME = 'RADY' AND FIRSTNME = 'BOB'
```

If either the first or second portion of the compound predicate evaluates as false then the entire row evaluates as false. The following is an example of a non-Boolean term compound predicate.

```
WHERE LASTNAME = 'RADY' OR FIRSTNME = 'BOB'
```

In this situation DB2 must look at the second predicate even if the first evaluates as false. Using non-Boolean term predicates in queries can limit DB2's use of indexes. In the case of the "OR" predicate above, DB2 can at best use multi-index access.

While non-Boolean term predicates can still present a performance challenge, there are improvements to DB2's handling of certain previously non-Boolean term predicates. DB2 11 for z/OS is able to convert certain non-Boolean term predicates involving IN's and OR's to index matching predicates for situations including "IS NULL" predicates. In addition, DB2 11 for z/OS has improved index matching capabilities for some IN/OR combinations to deliver multi-index access where such access was previously not possible. This is accomplished via improvements to query transformation as well as access path selection. For example, the following query could at best get multi-index access in previous versions of DB2 for z/OS:

```
SELECT LASTNAME
FROM   EMP
WHERE  WORKDEPT = ? OR WORKDEPT IS NULL
```

Under DB2 11 for z/OS the compound predicate in this query is transformed into an IN predicate and can get IN-List matching index access. The transformed query would look something like this:

```
SELECT LASTNAME
FROM   EMP
WHERE  WORKDEPT IN (?,NULL)
```

A query with an IN-List and non-Boolean term predicate testing optionality on a column can also be transformed into IN-LIST table and nested loop join. This was an access path that was introduced in DB2 10 for z/OS for multi-element IN-Lists but has been expanded now to handle this "OR IS NULL" condition. For example, the following query:

```
SELECT LASTNAME
FROM   EMP
WHERE  WORKDEPT IN (?,?) OR WORKDEPT IS NULL
```

This example would at best get a non-matching index scan in previous versions of DB2 for z/OS. However, in DB2 11 for z/OS all three values will be combined into an IN-List to support matching index access.

A query containing an inequality with an “OR IS NULL” condition could at best get multi-index access in previous versions of DB2 for z/OS.

```
SELECT LASTNAME
FROM   EMP
WHERE  WORKDEPT > ? OR WORKDEPT IS NULL
```

In DB2 11 for z/OS this query can now get single column matching index access.

Some complex search queries or queries that are generated by various tools may end up with combinations of IN and OR predicates. These predicates at best in previous versions of DB2 for z/OS were eligible for non-matching index access.

```
SELECT LASTNAME
FROM   EMP
WHERE  WORKDEPT IN (?,?,?) OR WORKDEPT = ?
```

In DB2 11 for z/OS this query could be rewritten as multiple OR’s and thus be eligible for multi-index access matching on a combination of the IN-List and equality predicates.

PREDICATE PUSHDOWN INTO MATERIALIZED VIEWS AND TABLE EXPRESSIONS

Complex SQL is important, and the usage of complex SQL is encouraged especially when one single complex statement can replace several simple statements. Many DB2 customers are using complex SQL statements for report writing as well as some transaction processing. In addition, customers use tools such as ERP applications and reporting writing software to create complex reports quickly and easily. One SQL construct that adds a lot of power to the language is nested table expressions. Since the output from SQL is a table with columns, nested table expressions, along with views, have been used quite extensively to build one complex process over another in a single SQL statement. When processing a nested table expression or view the DB2 engine makes a decision as to whether or not the nested table expression or view is merged with the outer portion of the statement that references it, or if the nested table expression or view is materialized in a work file before further processing. The SQL Performance Guide clearly documents when a nested table expression or view is materialized, but some simple examples are where the nested table expression or view contains a GROUP BY (most times) or a DISTINCT. In previous versions of DB2 the materialization of nested table expressions and views presented a challenge when predicates were coded against (outside or after) the table expression or view. In some limited cases DB2 was able to push the predicates down into the view or table expressions, but not when the view or table expression was materialized. This resulted in complex queries with materialized views and table expressions reading entire sets of data into a work file only to have some or most of that data filtered later on in the processing of the query. The end result was reporting queries that used large amounts of resources to produce relatively small result sets.

DB11 for z/OS improves the processing of these types of queries by improving its ability to push predicates into materialized table expressions and views during the query transformation process. This enables the optimizer to filter predicates earlier during processing and seriously reduce work file access and row processing which can result in dramatic reduction in elapsed time and CPU for these types of queries. This pushdown does not happen all of the time, but can happen for certain predicate constructs that are common in some statements that are generated by popular tools. However, in the situations where predicates are pushed down into the materialized view or table expression the performance advantage can be significant. This is because the filtering can be applied before materialization happens, and that can avoid the data for filtered rows from being access twice before filtering. In addition, by pushing the predicates into the table expression or view DB2 may be able to take advantage of indexes on tables inside the table expressions or views for further performance improvement. The types of predicates that can be pushed down include:

- **Predicates in an ON clause**

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
LEFT OUTER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP GROUP BY WORKDEPT
) AS B(WORKDEPT, CNT)
ON A.WORKDEPT = B.WORKDEPT
AND   B.WORKDEPT = 'C01';
```

In DB2 11 query transformation will now generate a predicate inside the table expression, and the rewritten query which can be viewed using IBM Data Studio or by looking in the DSN_QUERY_TABLE would look something like this (for either an inner or outer join):

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
LEFT OUTER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP GROUP BY WORKDEPT
WHERE  EMP.WORKDEPT = 'C01'   < - - generated predicate example>
) AS B(WORKDEPT, CNT)
ON A.WORKDEPT = B.WORKDEPT
AND   B.WORKDEPT = 'C01';
```

Another place that the generated predicate will appear is in the DSN_PREDICAT_TABLE EXPLAIN table.

- **Non-Boolean term OR predicates**

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
INNER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP GROUP BY WORKDEPT
) AS B(WORKDEPT, CNT)
ON     A.WORKDEPT = B.WORKDEPT
WHERE B.WORKDEPT LIKE 'C%' OR B.WORKDEPT LIKE 'A%'
```

Prior to DB2 11 these types of predicates would be pushed down if they were Boolean term only (not having the “OR”). Beginning with DB2 11 these types non-Boolean term predicates can be pushed down into the table expression (or view). The transformed query will look something like this:

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
INNER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP
WHERE EMP.WORKDEPT LIKE 'C%' OR EMP.WORKDEPT LIKE 'A%'
GROUP BY WORKDEPT
) AS B(WORKDEPT, CNT)
ON     A.WORKDEPT = B.WORKDEPT
WHERE B.WORKDEPT LIKE 'C%' OR B.WORKDEPT LIKE 'A%'
```

- **Stage 2 predicates**
  It is quite common for some tools to generate queries that apply functions to columns in tables, but also against views and table expressions. DB2 11 can apply these stage 2 predicates inside the materialized views and table expressions.

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
INNER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP GROUP BY WORKDEPT
) AS B(WORKDEPT, CNT)
ON     A.WORKDEPT = B.WORKDEPT
WHERE UPPER(B.WORKDEPT) = 'C01'
```

Can be transformed to this in DB2 11:

```
SELECT EMPNO, SALARY, CNT
FROM   EMP A
INNER JOIN
(
SELECT WORKDEPT, COUNT(*)
FROM   EMP GROUP BY WORKDEPT
WHERE  UCASE(EMP.WORKDEPT) = 'C01'
) AS B(WORKDEPT, CNT)
ON     A.WORKDEPT = B.WORKDEPT
WHERE UPPER(B.WORKDEPT) = 'C01'
```

These enhancements come without any programming changes and will take affect once a statement is recompiled (prepared or bound) under DB2 11. There are still some limitations such as predicates that are generated as part of predicate transitive closure will not be pushed into a materialized view or table expression, but they are for merged views and table expressions. Also, any stage 2 predicates that are expressions pushed down into a materialized view or table expression that could take advantage of an index on expression will not. While your results may vary the bottom line is that these types of queries can experience significant performance improvements in DB2 11 versus previous releases of DB2.

PRUNING OF ALWAYS TRUE AND ALWAYS FALSE PREDICATES

Query performance is always a concern of management once an application has been implemented. However, quite honestly, managements' concerns during application development are not of performance but rather speed of delivery and flexibility of the code. They want highly adaptable applications delivered quickly. Yet, they are always surprised when performance is less than perfect once the application is implemented. Typical discoveries during performance analysis of implemented applications reveal SQL statements that are highly adaptable and few in numbers. That is, during application development the developers took it upon themselves to produce SQL statements that could adapt to a variety of inputs, typically to back processes such as generic search screens. This reduced the quantity of code required and greatly improved the delivery time and flexibility of the code.

The downside to this type of programming meant that at statement compile time (prepare for dynamic or bind for static) DB2 had to choose indexes and table access sequence based upon a combination of predicates that may or may not be effective at execution time. It is quite common that this flexible programming technique contains predicates that may be always true or always false during statement execution. Since DB2 has chosen an access path at statement compile and not statement execution then predicates that are always false at execution will be used to determine access path during compile and will be evaluated during execution. The same is true for some predicates that are always true as well. While DB2 did have the ability to prune some always false predicates in the past there were still situations, especially in the case of the compound predicates, where the pruning did not take place. In many cases improving the performance of these types of queries required that the application developer split the query into multiple separate queries that were executed depending upon program logic, or to split the query internally using UNION ALL to divide the compound conditions and get the desired access paths for each block of the UNION ALL. There may also be other situations where the flexible SQL is generated by a tool and the developer has no control over that SQL. The solution in these situations prior to DB2 11 usually required severe creative skills by the DBA to squeeze every ounce of performance out of DB2.

DB2 11 extends the pruning of always true and always false predicates, and thus greatly reduces the need for program changes in support of high performance generic search queries. For always true predicates, such as "AND 1=1", DB2 will prune this predicate so that it is not evaluated during statement execution. This is handled by the query transformation component of the DB2 engine. For example, the following query:

```
SELECT LASTNAME
FROM   EMP
WHERE  EMPNO = '000010'
AND    1=1
```

Becomes:

```
SELECT LASTNAME
FROM   EMP
WHERE  EMPNO = '000010'
```

While prior to DB2 11 query transformation was able to prune some always false predicates such as this:

```
SELECT LASTNAME
FROM   EMP
WHERE  EMPNO = '000010'
OR     'A' = 'B'
```

it was not able to prune always false situations for compound predicates. This changes in DB2 11 in that DB2 can prune some compound always false predicates. This is true only for dynamic SQL that contains literal values, which is the case for a lot of SQL created by application development tools. Imagine, for example, an application process that provides a generic search on the employee sample table. The screen provides for a search on employee number and/or employee date of birth. The database has an index on both columns, and the statement is written something like this for a search by employee number:

```
SELECT LASTNAME
FROM   EMP
WHERE  (EMPNO = '000010'
        AND    1=1)
OR     (BIRTHDATE = '1954-01-01'
        AND    1=2)
```

Prior to DB2 11 the best access path for this query was likely multi-index access given an index on both EMPNO and BIRTHDATE. In fact, that would likely be the access path regardless of the input entered. Beginning with DB2 11, query transformation will prune all the unnecessary bits in the compound predicate. Thus the statement above becomes this:

```
SELECT LASTNAME
FROM   .EMP
WHERE  EMPNO = '000010'
```

This enables access path selection to choose single matching index access in support of the predicate. Likewise, if the input query is for a search on the employee date of birth only:

```
SELECT LASTNAME
FROM   EMP
WHERE  (EMPNO = '000010'
        AND    1=2)
OR     (BIRTHDATE = '1954-01-01'
        AND    1=1)
```

Then query transformation will prune the predicates such that the statement that is sent on to access path selection looks like this:

```
SELECT LASTNAME
FROM   DANL.EMP
WHERE  BIRTHDATE = '1954-01-01'
```

This enables access path select to choose single matching index access in support of the predicate on BIRTHDATE is such an index exists.

This enhancement to query transformation in DB2 11 can be an incredible boost to these types of compound generic search queries. Keep in mind, however, that this will only work for dynamic SQL with embedded literal values and not for host variables/parameter markers or REOPT. Also, keep in mind that the always false predicate of “OR 0=1” is never pruned as this predicate has been used as a query tuning technique to disable indexability of other predicates for many years.

ANALYTICS ENHANCEMENTS

In this part of the DB2 11 White Paper we are going to cover the new analytics features provided with DB2 11. Some of these features have also been retrofitted back into DB2 10 and as such may not be considered unique to DB2 11, but we are anyway going to cover these features, although perhaps with less depth compared to what are exclusive for DB2 11.

In summary this is what we are going to cover in this section.

- DB2 11 features rolled back to DB2 10
  - DB2 support for IDAA (IBM DB2 Analytics Accelerator) V3
  - High performance SPSS in-transaction scoring via PACK/UNPACK
- New features unique to DB2 11
  - SQL Grouping Sets, Rollup, and Cube
  - IFI 306 performance improvement for CDC (Change Data Capture) and IDAA v3
  - Hadoop access via table UDF

Some of these features will require enabling functions from other products to provide actual benefit, which will be highlighted as specifics topics are covered.

Note that there are several other features discussed within this paper that are also related to analytics such as:

- Temporal Enhancements on page 21
- Transparent Archive Query on page 56

DB2 SUPPORT FOR IDAA V3

The integration between DB2 for z/OS and IDAA (IBM DB2 Analytics Accelerator) V3 is a central corner piece in IBM's strategy to position DB2 and z/OS as a viable analytics platform with some unique offerings. Obviously this feature only brings added value when the Netezza is part of the mainframe hardware estate.

David Barnes, who in his keynote speech at IDUG in Berlin 2012 explained his 90/90 percent rule, saying that although 90% of the world's data may be unstructured, the 90% of the data that is important to us is already well taken care of, by people like you and I, today's data management practitioners and tomorrow's data scientists. He also explained that Big Data will not necessary provide any answers, but teach us to ask the right questions towards the data we already know and have securely stored and managed using DB2 for z/OS.

Hence, as we efficiently organize the data we already have and take into account Big Data type feeds the vicinity to data becomes important and having hybrid technologies like DB2 and IDAA fully integrated provides a very promising outlook allowing for consolidation and unification of transactional and analytical data stores.

Augmentation of existing structured data with new insight extracted from Big Data will allow us to formulate the right questions and create new business opportunities.

WHAT IS NEW WITH IDAA V3?

Before we move on to look at some of the new use cases made feasible with IDAA V3 lets quickly refresh what the new features are.

With the hardware refresh of the Netezza hardware platform one can now store 1.3 PB of data, which should be sufficient for the most capacity needs.

Using Change Data Capture, where IFI 306 READs are be used to read log records via IFCID 306, it is now possible to have changes to DB2 data propagated to the accelerator as they happen. This is a near real-time replication mechanism that will ensure that queries redirected to the accelerator are delivering close to real time results. To minimize overhead IFI filtering has been introduced as a performance improvement for CDC.

In cases where data changes are not being propagated it is a challenge to manage data freshness and consequently the accuracy of query results. Therefore a mechanism has been introduced for detecting staleness of data by exploiting the RTS last change timestamp column. This allows for finding out if DB2 data for a table or partition has changed since last load onto the accelerator and avoid refreshing data that is already up-to-date without requiring explicit knowledge about data changes.

Previously it was mandatory to keep data both in DB2 as well as in the accelerator, which would tend to drive up the storage cost for IDAA. Using an IBM provided Stored Procedure individual partitions can now be migrated to the accelerator and after completion and having secured the data with a partition level image copy then the partition is emptied and space is released. This feature is referred to as HPSS (High Performance Storage Saver).

Directing queries to DB2 or IDAA can be controlled either transparently via DB2 subsystem parameter GET_ACCEL_ARCHIVE or explicitly using the special register CURRENT GET_ACCEL_ARCHIVE at the application level. The query will have to execute entirely in one of the two places, so if any rows required to satisfy the query reside in the accelerator the whole query will be executed there (i.e. partitions not migrated also reside on IDAA). Restrictions that apply for HPSS are that the table must reside in a range partitioned tablespace and it cannot be part of any RI constraints.

Previously, when enabling query acceleration the actual outcome would be depending on access path cost estimation and some heuristics criteria. In order to increase the scope of query off-load two new values have been introduced for the special register CURRENT QUERY ACCELERATION (default value inherited from ZPARM QUERY_ACCELERATION). When set to ELIGIBLE a query is routed to the accelerator if it satisfies the acceleration criteria irrespective of the cost and heuristics criteria. Otherwise, it is executed in DB2. The option ALL means a query is always routed to the accelerator. However, if it cannot be executed there, the query fails and a negative return code is passed back to the application.

The determination of eligibility has to do with the fact that IDAA does not support all features of DB2 SQL and as such successively we will see more and more queries becoming eligible as functions are being introduced in Netezza, the underlying DBMS in IDAA. Following is a short list of features added in V3:

- Byte-based string functions on data encoded by multi-byte character sets (like Unicode), but results will only be consistent with DB2 for single-byte characters (QUERY_ACCEL_OPTIONS must have value 3 to enable this).
- Character-based string functions like SUBSTRING, LOCATE, etc. with CODEUNITS32 in the list of arguments.
- Character-based string functions with CODEUNITS16 or OCTETS in argument will only be offloaded when QUERY_ACCEL_OPTIONS contains option 3.
- Additional OLAP functions for moving aggregates, like AVG, CORRELATION, COUNT, COUNT_BIG, MAX, MIN, STDDEV, SUM, VARIANCE, and VARIANCE_SAMP.

Workload Manager Integration has been added to support workload isolation and query prioritization. This helps to ensure that designated DB2 subsystems have a guaranteed amount of available resources on a shared accelerator and that the DB2 WLM service class importance level is honoured in the accelerator as well.

Monitoring and control capabilities have been improved in the following areas:

- IFCID 106 to report new ZPARMs and values
- Stats collection in DB2 for the accelerator related activities on stats interval instead of thread termination
- Account for the thread wait time on the event of query acceleration.
- IFCID 148 for thread activity now reflects if the thread is currently in query acceleration processing

New optimizations have been added to make data available for analytics faster. The refreshing of tables or partitions is now much faster and less resources intensive by use of an optimized version of unloading data from DB2. This is done by enhancing IDAA to understand DB2 internal formats, so the data can simply be unloaded without performing any transformation. Required data and format transformation is subsequently done highly parallelized on the accelerator.

NEW ENHANCEMENTS ARE MAKING MORE USE CASES FEASIBLE

The obvious use case for IDAA is where the sheer speed of MPP enables the business to apply analytics to massive amounts of data in order to create answers in terms of hours or even minutes instead of days. However, in many cases organizations already have these capabilities available today using other types of technologies running off the mainframe, and even though time to deliver data could be improved having processing power closer to data this may not be sufficient to change current practices due to the significant amount of investments already done in MPP elsewhere.

As such the fundamental success criteria in IBM renewed focus on the mainframe playing a key role in business intelligence and analytics is the tight integration aspect of the equation.

The incremental update in IDAA as one such key integration feature, which in essence extends the accelerator use to reporting on operational data delivering near real-time results as the OLTP systems continuously are changing the state of the business. It also improves efficiency and availability as snapshot refresh of data is no longer required.

Depending on current design of handling current versus historical data the High Performance Storage Saver also provides some opportunities for implementing sophisticated multi-temperature data solution, significantly reducing the cost for the storage resources. In particular this is true for use cases where the partitioning scheme used for existing table designs are along the time dimension of data actuality.

As Netezza is being taught to understand more and more of the DB2 for z/OS SQL dialect the eligibility of queries to take advantage of the lightning speed of MPP will increase. Not only will it upsurge the number of business relevant use cases, but it will also protect the investment in hybrid computing technologies like IDAA.

WHAT IS IN THE PIPELINE FOR THE NEXT VERSION?

As we look at the list of some of the things that are in the pipeline it is clear that there is a focus on increasing the offload capability making more queries eligible for acceleration:

- Support of static SQL
- Qualitative predicates
- More built-in functions (e.g. BITAND)
- More scalar functions
- DB2 bi-temporal support

Beyond V3, there is the intention to work towards allowing partial offload of a query to the accelerator, which will be particularly valuable for use cases where current and history is separated by physical table boundaries. For mainly operational reasons in terms of manageability and recoverability there are applications designed and built in such a way that considering using IDAA for the historical data will mandate such a sub-query acceleration capability.

SQL GROUPING SETS, ROLLUP, AND CUBE

With DB2 10 the first wave of analytics SQL extensions were introduced into DB2 for z/OS, providing the RANK() and ROW_NUMBER() functions as well as moving aggregate windows where standard aggregate functions and advanced statistical functions could be applied (i.e. the OLAP specification). In essence this was a very strong statement from IBM in terms of putting actual development dollars into their z/OS based RDBMS engine in support of their strategy to position the mainframe as a viable platform for data warehousing and analytics processing.

Now with DB2 11 taking it one step further by also providing the GROUPING SETS, ROLLUP and CUBE capabilities Silicon Valley Lab has now completed provided a very compelling set of analytics features as part of the base product.

The new constructs are extensions to the GROUP BY clause in order to support so-called Super Groups and can be summarized as follows:

- The extensions GROUPING SETS is fundamental building block for GROUP BY operations and allows multiple grouping clauses to be specified in a single statement.
- ROLLUP is where values are being aggregated along a dimension hierarchy, like for instance week number, weekday, sales person. In addition to aggregation along the three dimensions a grand total is also being produced. There can be multiple such ROLLUPs in a single query in order to produce a result set containing multidimensional hierarchies.
- CUBE is where data is being aggregated based on columns from multiple dimensions and as such is equivalent to a cross tabulation, or better to relate it to ROLLUP we are talking about multiple rollups of height one.

SO WHAT EXACTLY DOES GROUPING SETS DO FOR ME?

Whereas the GROUP BY clause only allow us to group and aggregate data one way in a single query the extension with the GROUPING SETS option provides us the capability of specifying multiple grouping criteria being processed in one single pass of the data.

As an illustration let's consider what would happen if we executed the following query:

```
Select a, b, c, Sum(d)
  From MyTable
  Group By Grouping Sets (a, b, c);
```

The cardinality of the result table from this query would be the sum of distinct values across the three columns a, b, and c and each resulting row would have a distinct value from each of the columns respectively with null values for the other two and then the aggregated sum of column d.

To be a little more concrete here is an example using the sample application "Having Fun with Analytics on DB2 for z/OS", which can be download from idug.org, including DDL, data generating queries and sample queries covered in this white paper, plus more. Go check it out!
http://www.idug.org/p/do/sd/sid=5626&fid=4348&req=direct

Given the Customer and Transaction table we execute the following query to produce grouping sets for Customer Type, City Name, and Transaction Text:

```
Select Value(c.CustTp,'@SubTotal')  As "CustTp"
     , Value(c.CityNm,'@SubTotal')  As "CityNm"
     , Value(t.TransTx,'@SubTotal') As "TransTx"
     , Sum(t.TransAm)               As "TransAm"
  From TFwaCust c
 Inner Join TFwaTrans t
```

```
     On c.CustId = t.CustId
   Group By Grouping Sets (c.CustTp, c.CityNm, t.TransTx);
```

As a result based on having sized the Customer table with 1000 rows and the Transaction table with 50K rows, we see something like this (here only showing rows with significant changes in aggregation):

```
---------+---------+---------+---------+---------+---------+---------+
CustTp     CityNm             TransTx                     TransAm
---------+---------+---------+---------+---------+---------+---------+
ANY        @SubTotal          @SubTotal               48821618.73
......
YYC        @SubTotal          @SubTotal                1032142.15
@SubTotal  Aarau              @SubTotal                3209198.53
......
@SubTotal  Zürich             @SubTotal                2020306.39
@SubTotal  @SubTotal          Books                    5559682.30
......
@SubTotal  @SubTotal          Women's Shoes           19697450.57
DSNE610I NUMBER OF ROWS DISPLAYED IS 54
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+
```

SO WHAT EXACTLY DOES ROLLUP DO FOR ME?

The ROLLUP extension to the GROUP BY clause allows us to do aggregation within a hierarchy of columns. When defining the hierarchy it is important to keep in mind that the sequence of the columns is significant, where the hierarchy top to bottom is defined by the sequence of columns left to right.

As an illustration let's consider what would happen if we executed the following query:

```
Select a, b, c, Sum(d)
  From MyTable
  Group By Rollup (a, b, c);
```

This would produce sums for each distinct set of values at each of the four levels in the hierarchy (i.e. including the grand total). As such it would be the same as if we had done the combination of Group By (a), Group By (a, b), Group By (a, b, c), and the grand total. For columns not relevant for a given summation null vales are returned.

Here is another example using the sample application "Having Fun with Analytics on DB2 for z/OS" referred to earlier.

Given the Customer and Transaction table we execute the following query to produce rollups with aggregation along the hierarchy top down GrandTotal, Customer Type, Sales Year, and Sales Month:

```
Select Value(c.CustTp,'@SubTotal')                          As "CustTp"
    , Value(Char(Year(t.TransTs)),'@SubTotal')              As "Year"
    , Value(Substr(Digits(Month(t.TransTs)),9),'@SubTotal') As "WeekNo"
```

```
   , Sum(t.TransAm)                                         As "TransAm"
  From TFwaCust c
 Inner Join TFwaTrans t
    On c.CustId = t.CustId
  Group By Rollup (c.CustTp, Char(Year(t.TransTs))
                          , Substr(Digits(Month(t.TransTs)),9));
```

As a result based on having sized the Customer table with 1000 rows and the Transaction table with 50K rows, we see something like this (here only showing rows with significant changes in aggregation):

```
---------+---------+---------+---------+---------+---------+---------+
CustTp     Year         WeekNo                           TransAm
---------+---------+---------+---------+---------+---------+---------+
ANY        1992         09                                985.82
......
ANY        2012         10                                308.05
BYC        1993         04                                981.98
......
YYC        2004         05                                  4.62
ANY        1992         @SubTotal                        1970.52
......
ANY        2012         @SubTotal                       20676.58
BYC        1993         @SubTotal                       11078.85
......
YYC        2004         @SubTotal                         317.21
ANY        @SubTotal    @SubTotal                     48821618.73
......
YYC        @SubTotal    @SubTotal                      1032142.15
@SubTotal  @SubTotal    @SubTotal                     55414230.01
DSNE610I NUMBER OF ROWS DISPLAYED IS 829
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+
```

Important to notice is that the sequence of columns in the GROUP BY ROLLUP clause is significant and we need to keep in mind the purpose of the analysis when designing the query. Had we for instance specified the three parts in reverse order then we would have seen partial rollups by Month and Year, as well as Month.

SO WHAT EXACTLY DOES CUBE DO FOR ME?

The CUBE extension to the GROUP BY clause allows us to do a complete cross tabulation across a set of column dimensions. Contrary to rollup, for cube the sequence of columns is insignificant, since all permutations of column sets are re included in the calculations.

As an illustration let’s consider what would happen if we executed the following query:

```
Select a, b, c, Sum(d)
  From MyTable
  Group By Cube (a, b, c);
```

This would produce sums for each permutation of column sets, including the grand total). As such it would be the same as if we had done the combination of Group By (a), Group By (b), Group By (c), Group By (a, b) , Group By (a, c) , Group By (b, c), Group By (a, b, c), and the grand total. For columns not relevant for a given summation null vales are returned.

Here is another example using the sample application “Having Fun with Analytics on DB2 for z/OS” referred to earlier.

Given the Customer and Transaction table we execute the following query to produce cube with aggregation along the the three dimensions Customer Type, City, and Transaction Text:

```
With Cube As (Select Value(c.CustTp,'@SubTotal')  As CustTp
                   , Value(c.CityNm,'@SubTotal')  As CityNm
                   , Value(t.TransTx,'@SubTotal') As TransTx
                   , Sum(t.TransAm)               As TransAm
  From TFwaCust c Inner Join TFwaTrans t On c.CustId = t.CustId
 Group By Cube (c.CustTp, c.CityNm, t.TransTx))
Select Cell.CustTp As "CustTp" , Cell.CityNm  As "CityNm"
  , Cell.TransTx   As "TransTx", Cell.TransAm As "TransAm"
  , Dec(Cell.TransAm,11,2)*100/Dec(CustTp.TransAm,11,2)  As "CustTp%"
  , Dec(Cell.TransAm,11,2)*100/Dec(CityNm.TransAm,11,2)  As "CityNm%"
  , Dec(Cell.TransAm,11,2)*100/Dec(TransTx.TransAm,11,2) As "TransTx%"
  , Dec(Cell.TransAm,11,2)*100/Dec(Grand.TransAm,11,2)   As "Grand%"
  From Cube Cell
  Join Cube CustTp On CustTp.CustTp = '@SubTotal'
   And Cell.CityNm = CustTp.CityNm And Cell.TransTx = CustTp.TransTx
  Join Cube CityNm On CityNm.CityNm = '@SubTotal'
   And Cell.CustTp = CityNm.CustTp And Cell.TransTx = CityNm.TransTx
  Join Cube TransTx On TransTx.TransTx = '@SubTotal'
   And Cell.CityNm = TransTx.CityNm And Cell.CustTp = TransTx.CustTp
  Join Cube Grand On Grand.CustTp = '@SubTotal'
   And Grand.CityNm = '@SubTotal' And Grand.TransTx = '@SubTotal'
 Where Cell.CustTp <> '@SubTotal' And Cell.CityNm <> '@SubTotal'
   And Cell.TransTx <> '@SubTotal';
```

In order to better illustrate the result of the operation in a more condensed form we are joining the cube with itself in order to calculate percentages between the cell values and aggregated values for Customer Type, City, Transaction Text and Grand Total. It should be noted that we are in fact ignoring some rows in the cube where aggregation is done along two dimensions (i.e. Type and City, Type and Text, as well as City and Text).

As a result based on having sized the Customer table with 1000 rows and the Transaction table with 50K rows, we see something like this (here only showing rows with significant changes in aggregation):

```
---------+---------+---------+---------+---------+---------+---------+---------
CustTp CityNm      TransTx              TransAm CustTp% CityNm% TransTx% Grand%
---------+---------+---------+---------+---------+---------+---------+---------
ANY    Bern        Books              486699.36 100.000   9.791   13.856   .878
......
```

```
ANY    Biel/Bienne Books              224509.53 100.000    4.516   10.719   .405
BYC    Herisau     Books               39746.23  24.509   13.799    8.467   .071
......
YYC    Aarau       Books               70769.92  28.416   65.633    9.655   .127
ANY    Bern        Computer Equipment  49050.19 100.000    6.353    1.396   .088
......
YYC    Aarau       Computer Equipment  15813.01  36.506   66.386    2.157   .028
ANY    Bern        CD & MP3 Music     179995.59 100.000    7.639    5.124   .324
......
YYC    Aarau       Women's Shoes      312466.96  34.995   69.564   42.629   .563
DSNE610I NUMBER OF ROWS DISPLAYED IS 654
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------
```

Although the other examples have given the illusion that sequence of rows in the result is kind of meaningful along the dimensions of the grouping operation this is not a given in this case because of the joins taking place. As such if we need a certain order we should always follow good practices of having an ORDER BY clause.

As for the sequence columns in the GROUP BY CUBE clause it makes no difference. To illustrate this, the order of columns was reversed in the GROUP BY CUBE clause as part of adding an order by clause:

```
---------+---------+---------+---------+---------+---------+---------+---------
CustTp CityNm      TransTx                TransAm CustTp% CityNm% TransTx% Grand%
---------+---------+---------+---------+---------+---------+---------+---------
ANY    Baden       Books               94462.53 100.000    1.900    8.724   .170
......
ANY    Zürich      Books              294956.31 100.000    5.933   14.599   .532
BYC    Aarau       Books               70525.87  28.318   24.485   12.107   .127
......
YYC    Herisau     Books               37055.57  22.850   34.366   12.386   .066
ANY    Baden       CD & MP3 Music      45607.20 100.000    1.935    4.212   .082
......
YYC    Herisau     CD & MP3 Music      18603.81  19.905   34.085    6.218   .033
ANY    Baden       Computer Equipment  10550.20 100.000    1.366     .974   .019
......
YYC    Herisau     Women's Shoes      136706.78  20.117   30.435   45.696   .246
DSNE610I NUMBER OF ROWS DISPLAYED IS 654
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------
```

HADOOP ACCESS VIA TABLE UDF

DB2 is now providing the connectors and the database capability to allow DB2 applications to easily and efficiently access Hadoop data sources. This is done by introducing the generic table UDF capability in DB2 11, which enables the ability to produce a variable shape of UDF output table.

One such application is IBM Hadoop based BigInsights, which is IBM strategic Big Data platform. As such we see how DB2 11 bring yet another piece to the integration puzzle enabling traditional applications on DB2 for z/OS to access Big Data.

Analytic jobs are being specified using JSON Query Language (Jaql) and then submitted to BigInsights from with DB2 for z/OS. The result of the job is then stored in the Hadoop Distributed File System (HDFS).

Delivered with BigInsights there is a table UDF (HDFS_READ) that enable DB2 for z/OS applications to read the Big Data analytic result from HDFS, for subsequent use in an SQL query.

Some of the Big Data use cases being envisage with this new integration technology are things like:

- **Big Data Exploration**: Find, visualize, understand all big data to improve decision making.
- **Enhanced 360° View of the Customer**: Extend existing customer views (MDM, CRM, etc) by incorporating additional internal and external more unstructured information sources.
- **Security/Intelligence Extension**: Lower risk, detect fraud and monitor cyber security in real-time.
- **Operations Analysis**: Analyze a variety of machine data for improved business results.
- **Data Warehouse Augmentation**: Integrate big data and data warehouse capabilities to increase operational efficiency.

Again we see how essential the integration aspect is and how the future of large scale computing is going to be all about hybrid architectures. Going back to what David Barnes said in his IDUG Keynote, Big Data will not necessary provide any answers, but teach us to ask the right questions towards the data we already have in DB2 for z/OS.

GLOBAL VARIABLES

A long expected DB2 for z/OS feature, Global Variables, has finally arrived. Many application designers who are experienced with other relational databases were asking for this feature. Without Global Variables porting of many third-party applications to DB2 for z/OS is too difficult for a number of reasons:

- There is no easy way to share information between the SQL statements among applications. This creates a need for applications to put supporting logic inside their code to access and transfer data between SQL statements
- The security of the information being transferred is enforced by application logic.

Global Variables

- Enable the sharing of data between SQL statements without the need for application logic.
- Are maintained by DB2 and are available throughout the entire application scope.
- Have access controlled by GRANT and REVOKE statements.

SYNTAX AND SCOPE

- New CREATE VARIABLE statement, saved in DB2 catalog
- Associated with a specific application, value unique to the application
- The content is shared among the SQL statements within the same connection
- Similar to DB2 special registers
- With GRANT/REVOKE, privileges are assigned to Global Variables.

GLOBAL VARIABLE INSTANTIATION, REFERENCES AND MODIFICATIONS

- Instantiated upon the first reference.
- If created with the DEFAULT clause, the default expression is evaluated during instantiation

- If no DEFAULT is specified, NULL is used as the default value
- Can appear in expression, predicates, and select list.
- Cannot be used in, Check Constraints, MQTs, Definition expressions for indexes
- The value can be changed by SET, SELECT INTO, or VALUES INTO, an argument of an OUT or INOUT parameter in a CALL statement
- The content is not affected by COMMIT or ROLLBACK statements

GLOBAL VARIABLES WITH STORED PROCEDURE, UDF, TRIGGER

- The content of the Global Variables are always inherited from the caller.
- Stored Procedures can make changes to the Global Variables, but not if the SP was called by a UDF or trigger.
- The changes always propagate to the caller.

GLOBAL VARIABLES WITH THREAD-REUSE

- The content of the Global Variables persist across reusable threads.
- A reused thread keeps all values recorded from the previous thread.

GLOBAL VARIABLES AND DYNAMIC STATEMENT CACHE

- Runtime structures can be saved in the dynamic statement cache.
- Subsequent usage that match can skip the dynamic prepare process.
- Statement cache can be invalidated if the global variables used in the cached statement are DROPPED.

There are things we must be careful of while using Global Variables. Let’s say we have the following set up:

```
CREATE TABLE T1 (C1 SMALLINT,  C2 INTEGER)  IN GOLD123.TS1;
CREATE VARIABLE C1 INTEGER DEFAULT 1;
CREATE VARIABLE CX INTEGER DEFAULT 1;
COMMIT;
```

And table has 2 rows as below and Global Variables C1 and CX are initialized as above.

```
INSERT INTO T1 VALUES (1,111);
INSERT INTO T1 VALUES (5,222);
SELECT * FROM T1 WHERE C1= CX+4
```

Returns **(C1,C2) = (5,222)**

```
SELECT * FROM T1 WHERE C1= C1+4
```

Returns no rows!

C1 in C1+4 expression in WHERE clause is considered as Column Name, not Global Variable. If there are columns which has the same name with any Global Variables in the same SQL, those names will be resolved to column name. DB2 will follow the precedence rule to determine the most suitable match for an unqualified identifier. Column name is the first in the precedence rule.

SUMMARY

Core Benefits of Global Variables as general are as follows

- Porting competitor database vendor applications is faster and easier
- Cost of ownership for DB2 for z/OS is lowered
- Security is managed by DB2 and not application logic

VARIABLE ARRAYS

Until DB2 11, passing an array as a single IN/OUT parameter for a stored procedure was impossible. As a result, programmers would have to revert to using long list of parameters (if the number was not excessive), temp tables or passing the individual parameters in a long string.

Another new feature that DB2 11 brings in the area of variables is the ability to create an array data type. Arrays are a common and very useful construct in programming languages. It was direly needed in SQL/PL. Since SQL/PL adds procedural constructs to SQL and most stored procedures can be implemented entirely in SQL/PL, it is the next logical step to also bring the ability to use arrays to this programming language. DB2 11 allows to create two different types of arrays data types, the ordinary array and associative array.

The ordinary array is a construct that every programmer will be familiar with. It can be loosely defined as a number of data elements of the same type grouped under a single name (the array name). An individual element is addressed by combining the array name with an index. In the case of an ordinary array, this index is the ordinal number of the element. By definition, this index is of an integer type. In SQL/PL such an array is defined using the following syntax:

```
CREATE TYPE <array name>  AS <built-in type> ARRAY[<integer constant>]

Example: CREATE TYPE AMOUNTS AS DECIMAL(15,2) ARRAY[25]
```

An ordinary array thus has a fixed number of elements. The array elements can then be populated. It is probably wise to control the schema in which the array type will be created, just as you would do with any other user defined type.

The second type of array that can be defined is an associative array. With an associative array, an element is not addressed by its ordinal number but by a key that is 'associated' with it. As such, the array can be seen as a collection of key, value pairs.

Creating an associative array uses nearly the same syntax as an ordinary array. The difference is that instead of an integer constant defining the number of elements, the data type of the 'key' part is used. As a result, the associative array does not have a predefined cardinality.

Let me try and clarify the difference using a simple example. Consider the following two arrays:

1) Ordinary: CREATE TYPE DAYNAME AS VARCHAR(9) ARRAY[7]
2) Associative: CREATE TYPE DAYNUM AS SMALLINT ARRAY[VARCHAR (9)]

The arrays have the following sample contents:

'Monday'
'Tuesday'
'Wednesday'
'Thursday'
'Friday'
'Saturday'
'Sunday'

DAYNAME

"KEY"	"VALUE"
'Monday'	1
'Tuesday'	2
'Wednesday'	3
Thursday'	4
'Friday'	5
'Saturday'	6
'Sunday'	7

DAYNUM

Initializing an element in each using SQL/PL is done as follows:

Ordinary:	SET DAYNAME[3] = 'Wednesday';
Associative:	SET DAYNUM['Wednesday'] = 3;

Of course, using an ordinary array is easy. You can interrogate the value of each element by addressing the element using traditional methods. The index value is easy to determine since it must be an integer that is smaller than or equal to the cardinality of the array. A small sample could look like this:

```
CREATE TYPE NUMBERS AS INTEGER ARRAY[10];

CREATE PROCEDURE TEST01
   (IN  NUMARRAY  NUMBERS,
    OUT SUM       INTEGER)
  LANGUAGE SQL
  DISABLE DEBUG MODE
  MODIFIES SQL DATA
  DETERMINISTIC
  COMMIT ON RETURN YES

 BEGIN
     DECLARE I       INTEGER DEFAULT 1;
     SET SUM = 0;
     WHILE I <= CARDINALITY(NUMARRAY) DO
        SET SUM = SUM + 1;
     END WHILE;
 END
```

Note that in this example I have used a new function (CARDINALITY()) associated with array data types. It returns for ordinary arrays the maximum number of elements. For associative arrays it will return the number of actually initialized elements.

Accessing elements in an associative array may not always be so easy though, since you may not know what 'keys' have been assigned. DB2 11 provides a number of functions to support this:

ARRAY_FIRST(array)	Will return the minimum array index value of the argument. For an ordinary array this will typically be 1 (unless the array is empty). For an associative array, it will return the lowest assigned index value[4].
ARRAY_LAST(array)	Will return the highest array index value
ARRAY_NEXT(array,val)	Will return the next higher index value starting at val
ARRAY_PRIOR(array,val)	Will return the next lower index value starting at val

There is also a function, ARRAY_DELETE to remove unwanted elements from an array or even reinitialize an array altogether.

Besides these functions to work with arrays, another new function, UNNEST, allows the array to be integrated with SQL. The following example will deliver a result set not unlike the visual representation for DAYNUM (see above)

```
SELECT T.KEY, T.VALUE
  FROM UNNEST(DAYNUM) AS T(KEY; VALUE);
```

DB2 11 also provides a function, ARRRAY_AGG, that will allow you to populate an array with the result of an SQL statement. For instance, to fill an array with all the different employee numbers working on a project (using the IVP sample database):

```
CREATE TYPE EMPNO_AR AS CHAR(6) ARRAY[10];

CREATE PROCEDURE TEST01
   (IN  PROJECT   CHAR(6))

  LANGUAGE SQL
  DISABLE DEBUG MODE
  MODIFIES SQL DATA
  DETERMINISTIC
  COMMIT ON RETURN YES

 BEGIN
     DECLARE EMPNOS    EMPNO_AR;
     DECLARE I         INTEGER   DEFAULT 1;

     SET EMPNOS  = SELECT ARRAY_AGG(EMPNO)
                       FROM EMPPROJACT
                      WHERE PROJNO = PROJECT)
     WHILE I <= CARDINALITY(EMPNOS) DO
         Do something with  EMPNO[I];
     END WHILE;

 END
```

[4] In an associative array, the entries are kept in ascending order of the index value

While this is all very exciting, please note that with DB2 11, array data types can only be used in SQL PL (stored procedures and SQL scalar functions. Native SQL procedure can be called from distributed Java (JDBC, SQLJ) and C-based clients (ODBC, .NET).

DECLARED GLOBAL TEMPORARY TABLE ENHANCEMENTS

Some applications need to perform a number of different calculations on the same set of data. In many cases, the application can just define the result set and process the various calculations in sequence while processing the cursor that defines the set of data to work on. However, we have come across several situations where this is not possible. Of course, the application could just retrieve the data to work on repeatedly. If the retrieval of the data to work on is expensive, using a GTT can greatly reduce the cost and improve the performance of the overall process.

Among the reasons for using a GTT:

- The different calculations/processes work on different grouping levels of the data set.
- One calculation expands the data set and subsequent calculations need to process the expansion as well

Such applications would typically use like to use Declared GTT's but DGTT's are typically more expensive than CGTT because DGTT's do not have the ability to suppress logging. Further, DGTT's tend to generate more overhead cost due to incremental bind activity. So the application will be forced to make a balanced decision: Do I avoid the overhead associated with DGTT's and use CGTT or do I go for the ability to create an index on the GTT. Creation of an index is exclusive to DGTT.

We'll now describe how DB2 11 improves the situation.

NOT LOGGED DGTT

DB2 11 has made two major changes in the area of DGTT's that will improve the performance and cost of DGTT usage. The first improvement is the ability to define the DGTT as NOT LOGGED. Especially for DGTT that will be the target of high insert, update or delete activity, this brings a significant reduction in CPU cost. Also the volume of log data will be significantly reduced. All the log records that are required for UNDO processing are no longer generated. What remains is the small number of control records that are describe the creation and drop of the DGTT and some tablespace open/close records.

In order to drive this, the syntax to declare a DGTT has been enhanced. It now supports three options for logging:

1. LOGGED:

   This will be the default so that existing applications do not suddenly show a different behaviour. During ROLLBACK or ROLLBACK TO SAVEPOINT processing, changes to the DGTT are rolled back as well. Errors during SQL execution will undo the changes made to the DGTT contents

2. NOT LOGGED ON ROLLBACK PRESERVE ROWS

   The contents of the DGTT will not be changed during ROLLBACK or ROLLBACK TO SAVEPOINT. Any Errors during SQL insert, delete or update execution will result in deletion of the rows in the DGTT, the rows will NOT be preserved.

3. NOT LOGGED ON ROLLBACK DELETE ROWS

All the rows in the DGTT will be deleted during ROLLBACK or ROLLBACK TO SAVEPOINT processing if changes were made since the last commit. Errors during SQL execution will result in deletion of the rows in the DGTT.

Please note that if PRESERVE ROWS is specified for the ON ROLLBACK clause (or the ON COMMIT clause for that matter), the thread will not be eligible for reuse.

INCREMENTAL BIND / FULL PREPARE AVOIDANCE

The second improvement that is made to DGTT's deals with reducing the number of incremental binds and full prepares. This has been implemented by extending the scope of RELEASE DEALLOCATE. Previously, any statements that operate against a DGTT are not saved at commit. The obvious exception being an open cursor defined with hold. This behaviour was regardless of the RELEASE bind option of the package executing the SQL.

For static SQL this meant an incremental bind when the SQL statement was executed again. For dynamic SQL this meant another full prepare since dynamic SQL involving a DGTT is not eligible for the dynamic statement cache.

DB2 11 has changed the semantics of RELEASE DEALLOCATE to include SQL statements against DGTT's. As a result, incremental binds for static SQL against the DGTT will automatically be reduced if the package is bound with RELEASE DEALLOCATE. For dynamic SQL, the application will only benefit if the PREPARE of the statement is not repeated after a commit. No doubt this will require the programmer to revisit the code of applications that do commit while processing a DGTT since in DB2 V10, the PREPARE was lost and needed to be repeated after each commit. In DB2 11, provided the package is bound with RELEASE DEALLOCATE, the programmer just needs to perform the PREPARE once and subsequently only execute the prepared statement.

The more frequent an application commits, the bigger the savings of RELEASE DEALLOCATE will be with respect to DGTT processing.

Looking at all the other enhancements DB2 11 has brought to RELEASE DEALLOCATE, combined with enhancements in previous releases, RELEASE DEALLOCATE has become even more a valuable option in order to reduce cost.

SEE ALSO

Please see below for a cross reference to other topics that may also be of interest to application developers, but have been placed elsewhere within this document.

- Temporal Enhancements on page 21
- Online Schema Change Enhancements on page 24
- BIND/REBIND Enhancements on page 29
- INSERT Performance Improvements on page 32
- SQL Compatibility Feature on page 32
- XML Enhancements on page 35
- Utility Enhancements on page 41
- Security Enhancements on page 48

ACKNOWLEDGEMENTS

The IDUG DB2 11 Editorial Committee would like to thank the following people for their invaluable assistance in putting together this white paper:

- Akiko Hoshikawa
- Terrie Jacopi
- Jeff Josten
- Surekha Parekh
- Jim Pickel
- Terry Purcell
- Jim Reed