Sheryl M. Larsen
smlarsen@us.ibm.com

Sheryl Larsen is an internationally recognized researcher, consultant and lecturer, specializing in DB2 and is known for her extensive **expertise in SQL**. She co-authored a book, DB2 Answers, Osborne-McGraw-Hill, 1999.

She now works for IBM, but this material was developed before joining IBM and is the culmination of 25 years as an external consultant specializing in superhuman efforts to tune customer workloads.

Her new title is IBM's DB2 for z/OS Evangelist. Other titles include:

President of the Midwest Database Users Group,

IDUG Hall of Fame Speaker,

IDUG Hall of Fame Volunteer, and

Northern Illinois University Computer Science Alumni Council

Sheryl has over 25 years experience in DB2, has published articles, white papers, webcasts: bmc.com, ca.com, softbase.com, ibm.developerworks.com

Disclaimer/Trademarks

THE INFORMATION CONTAINED IN THIS DOCUMENT HAS NOT BEEN SUBMITTED TO ANY FORMAL IBM TEST AND IS DISTRIBUTED AS IS. THE USE OF THIS INFORMATION OR THE IMPLEMENTATION OF ANY OF THESE TECHNIQUES IS A CUSTOMER RESPONSIBILITY AND DEPENDS ON THE CUSTOMER'S ABILITY TO EVALUATE AND INTEGRATE THEM INTO THE CUSTOMER'S OPERATIONAL ENVIRONMENT. WHILE IBM MAY HAVE REVIEWED EACH ITEM FOR ACCURACY IN A SPECIFIC SITUATION, THERE IS NO GUARANTEE THAT THE SAME OR SIMILAR RESULTS WILL BE OBTAINED ELSEWHERE. ANYONE ATTEMPTING TO ADAPT THESE TECHNIQUES TO THEIR OWN ENVIRONMENTS DO SO AT THEIR OWN RISK.

ANY PERFORMANCE DATA CONTAINED IN THIS DOCUMENT WERE DETERMINED IN VARIOUS CONTROLLED LABORATORY ENVIRONMENTS AND ARE FOR REFERENCE PURPOSES ONLY. CUSTOMERS SHOULD NOT ADAPT THESE PERFORMANCE NUMBERS TO THEIR OWN ENVIRONMENTS AS SYSTEM PERFORMANCE STANDARDS. THE RESULTS THAT MAY BE OBTAINED IN OTHER OPERATING ENVIRONMENTS MAY VARY SIGNIFICANTLY. USERS OF THIS DOCUMENT SHOULD VERIFY THE APPLICABLE DATA FOR THEIR SPECIFIC ENVIRONMENT.

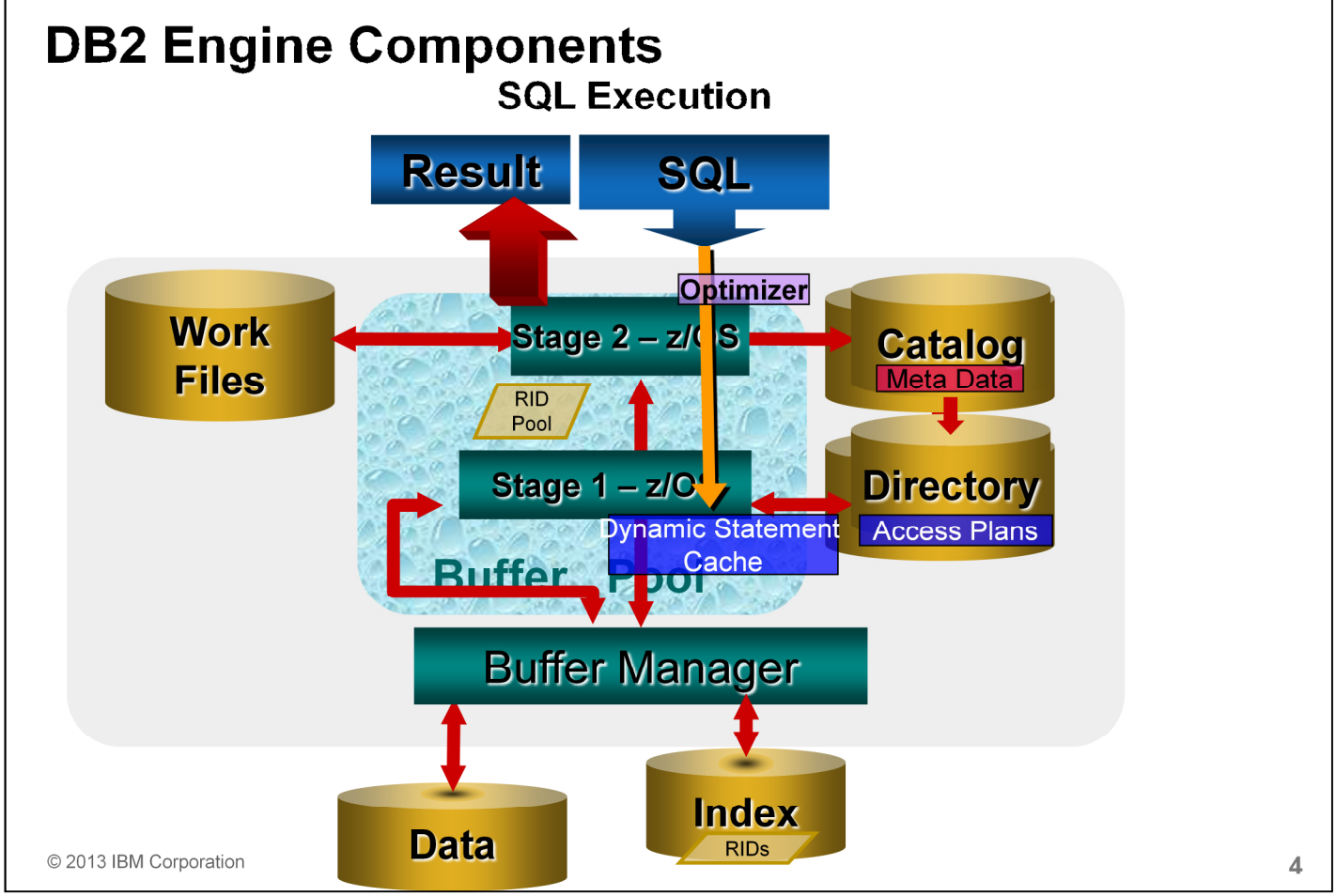
Trademarks
IBM, the IBM logo, ibm.com, and DB2 are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

IBM - A global computer products, services and solutions company based in the United States, IBM operates in more than 170 countries and has nearly 427,000 employees. With thousands of business partners and computer products suppliers worldwide, IBM generates more than USD99 billion in annual revenue. The company was founded in 1911 in Endicott, New York.

<slide 2>
This presentation draws on experiences both internal to IBM and at customer installations. There has not been any formal testing. The notices on this page explain this disclaimer and the trademarks commonly used in this presentation.

DB2 11 for z/OS Enhancements for Developers

- DB2 Engine Components
- Predicate Processing Intelligence
- Optimizer Details
- DB2 11 Performance Sweet Spots
- Easier DB2 Version Upgrade – application compatibility
- Big Data Capabilities
- Performance Improvements
  - no REBIND needed
  - REBIND required (with or without APRE
  - REBIND required (without APREUSE)
  - DBA or application effort required
- DDF Performance Improvements
- SQL changes/new features for DB2 V11
- Expanded Analytics Capabilities in DB2 11



For static SQL ,DB2 will use the stored access path in the Directory.

REOPT (ONCE), REOPT(AUTO) * DB2 9

For dynamic SQL ,DB2 will check the Dynamic Statement Cache for an exact match of the statement.

If found, the cached associated access path will be used.

REOPT(AUTO) *DB2 9

For dynamic SQL, DB2 will check the Dynamic Statement Cache for an exact match of the statement.

If found, the cached associated access path will be used.

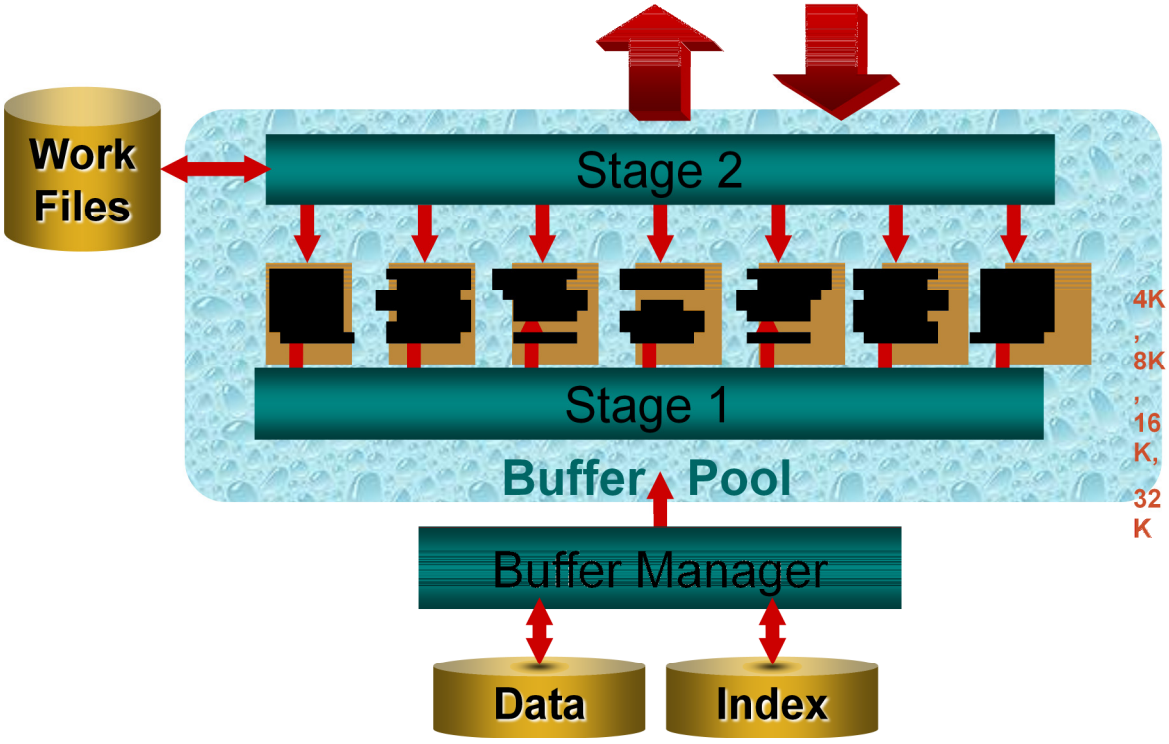
If found but the parameter marker values are not significantly different, the cached associated access path will be used.

If not found, the Optimizer costs out a new access path for use and stores it in the cache with the new statement

REOPT(Always)

For each execution, the Optimizer costs out a new access path for use ,

Page Processing – z/OS



Stage 1 filtering is done first against the 4K pages brought into the Buffer Pool.

Stage 2 only examines the rows that qualify after Stage 1 filtering, however, the entire row or index entry is still on the 4k page sitting in the Buffer Pool.

Once Stage 2 is complete, data transformations requested on the SELECT clause are performed prior to returning one result *value* at a time to the calling program.

Query response time is dependent on:

- The number of I/O's to pull data and/or index pages in the Buffer Pool
- The number of rows left after Stage 1 filtering
- The number of rows left after Stage 2 filtering
- The sequence the rows are in the Buffer Pool
- The amount of translations preformed on the result values

The less rows requested, the less columns requested, the less transformations, the faster the query goes.

Indexable Stage 1 Predicates

Predicate Type	Indexable	Stage 1
COL = value	Y	Y
COL = noncol expr	Y	Y
COL IS NULL	Y	Y
COL op value	Y	Y
COL op noncol expr	Y	Y
COL BETWEEN value1 AND value2	Y	Y
COL BETWEEN noncol expr 1 AND noncol expr 2	Y	Y
COL BETWEEN expr-1 AND expr-2	Y	Y
COL LIKE 'pattern'	Y	Y
COL IN (list)	Y	Y
COL IS NOT NULL	Y	Y
COL LIKE host variable	Y	Y
COL LIKE UPPER 'pattern'	Y	Y
COL LIKE UPPER 'host-variable'	Y	Y
T1.COL = T2.COL	Y	Y
T1.COL op T2.COL	Y	Y
T1.COL = T2 col expr	Y	Y
T1.COL op T2 col expr	Y	Y
COL=(noncor subq)	Y	Y
COL = ANY (noncor subq)	Y	Y
(COL1,...COLn) IN (noncor subq)	Y	Y
COL = ANY (cor subq)	Y	Y
COL IS NOT DISTINCT FROM value	Y	Y
COL IS NOT DISTINCT FROM noncol expr	Y	Y
T1.COL1 IS NOT DISTINCT FROM T2.COL2	Y	Y
T1.COL1 IS NOT DISTINCT FROM T2 col expr	Y	Y
COL IS NOT DISTINCT FROM (noncor subq)	Y	Y
value BETWEEN COL1 AND COL2	Y	Y
value BETWEEN col expr AND col expr	Y	Y
SUBSTR(COLX, 1, n) = value	Y	Y
DAY(TS_COL) = value	Y	Y
YEAR(DT_COL) = value	Y	Y
CASE WHEN THEN ELSE END = value	Y	Y

3
2

DB2 11 New Stage 1 Predicates

Stage 1 Predicates

Predicate Type	Indexable	Stage 1
COL <> value	N	Y
COL <> noncol expr	N	Y
COL NOT BETWEEN value1 AND value2	N	Y
COL NOT IN (list)	N	Y
COL NOT LIKE ' char'	N	Y
COL LIKE '%char'	N	Y
COL LIKE '_char'	N	Y
T1.COL <> T2 col expr	N	Y
COL op (noncor subq)	N	Y
COL op ANY (noncor subq)	N	Y
COL op ALL (noncor subq)	N	Y
COL IS DISTINCT FROM value	N	Y
COL IS DISTINCT FROM (noncor subq)	N	Y

1
3

43 TOTAL !!

Four Points of Filtering

1. Indexable = The predicate is a candidate for Matching Index access. When the optimizer chooses to use a predicate in the probe of the index, the condition is named Matching (matching the index). This is the first point that filtering is possible in DB2.
2. Stage 1 Index Screening = The Stage 1 predicate is a candidate for filtering on the index leaf pages. This is the second point of filtering in DB2.
3. Stage 1 Data Screening = The Stage 1 predicate is a candidate for filtering on the data pages. This is the third point of filtering in DB2.
4. Stage 2 = The predicate is not listed as Stage 1 and will be applied on the remaining qualifying pages from Stage 1. This is the fourth and final point of filtering in DB2.

DB2 11 New Indexable Stage 1 Predicates

- No longer worry about these:
  - WHERE value BETWEEN COL1 AND COL2
  - WHERE SUBSTR(COLX, 1, n) = value
  - WHERE SUBSTR(COLX, 1, n) *op* value
  - WHERE DATE(TS_COL) = value
  - WHERE DATE(TS_COL) *op* value
  - WHERE YEAR(DT_COL) = value
  - WHERE YEAR(DT_COL) *op* value

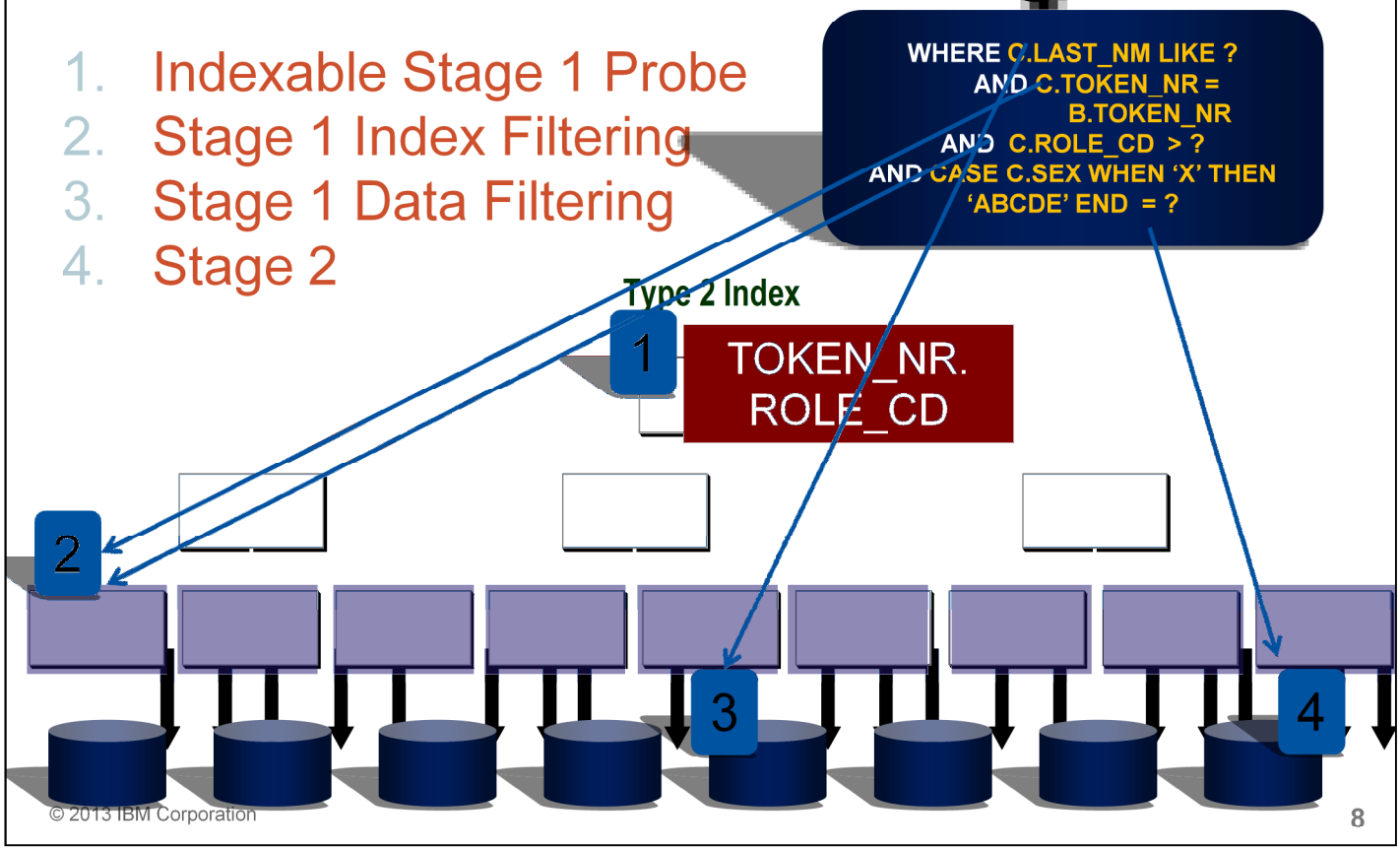
In the DB2 11 documentation **Summary of Predicate Processing**

  - WHERE value BETWEEN col expr AND col expr
  - WHERE CASE expr = value

The predicate ***might*** be indexable if *expr* contains one of the following scalar functions:
DATE
YEAR
SUBSTR (if the start value for the substring is 1.)

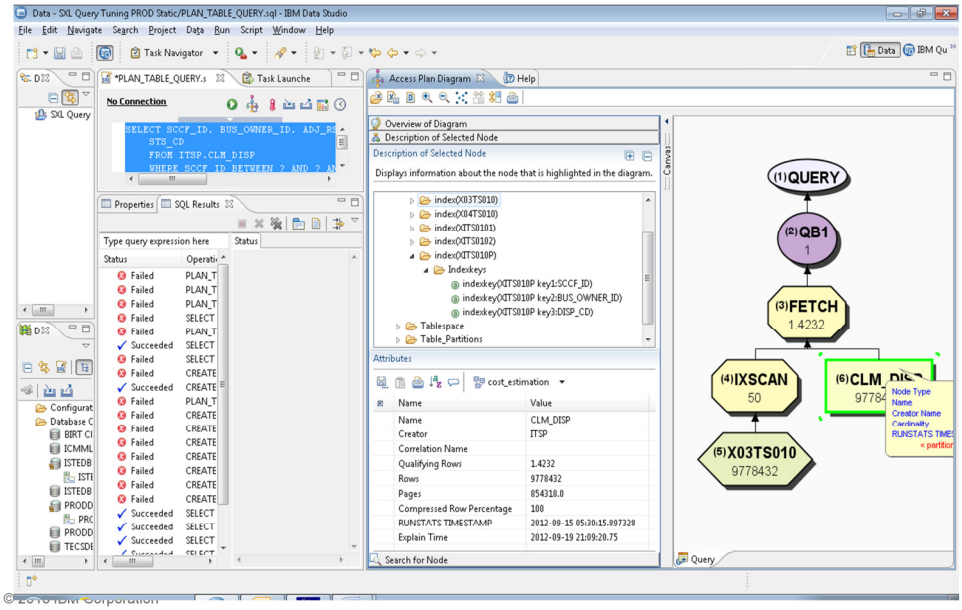
Mismatch data types also force the filtering to be held back to Stage 2. It is always a good idea in SQL coding to leave columns untouched and translate the other arguments. As query rewrite gets smarter over the years, this may not be necessary.

Four Points of Filtering – DB2

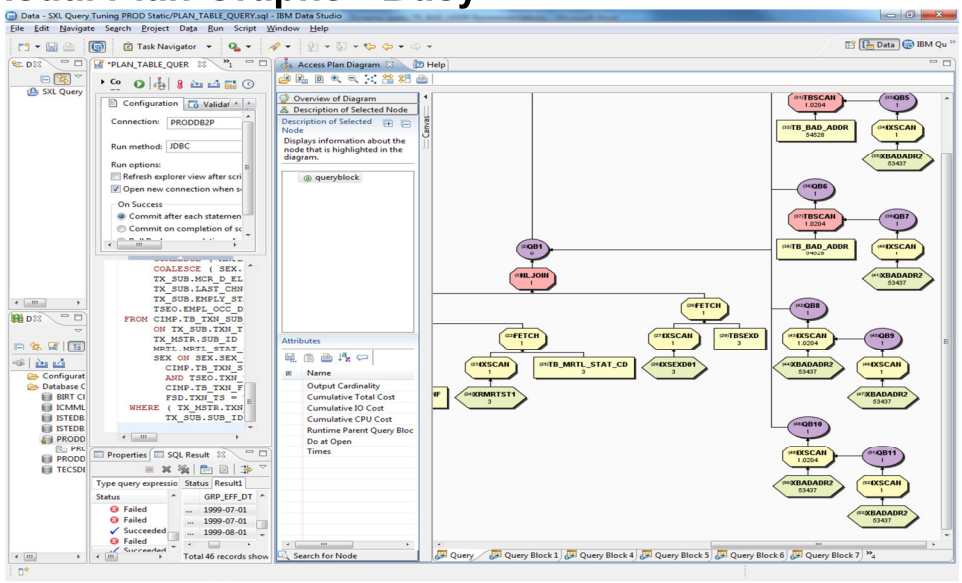


1. Indexable Stage 1 Probe – As of DB2 11, 32 predicates can be applied at this point. The ones that will be applied are dependent on which index was chosen, the conditions on the columns belonging in the index, and the sequence of those columns. If the first column of the index is used in a "=" predicate, the column is used to navigate the tree along with the next column (2 matching). If the next column is used in a "=" predicate, the column is used to navigate the tree along with the previous two columns (3 matching). If the next column is not an "=" predicate, the matching stops with this condition (4 matching) unless it is nonindexable or Stage 2 (3 matching). If the first column is not and "=" predicate, only the first column is used to navigate the tree (1 matching). The number of matching columns usually = one more than the last matching condition. Data types are required to match until V8.
2. Stage 1 Index Filtering - If there is no predicate involving the first column of the index, tree navigation is not allowed (0 matching). Any Stage 1 predicate (all 46) can be applied on the leaf page. This point of filtering is called index screening. Stage 2 conditions can also be applied after the Stage 1 conditions are applied (if this is index only access *and* the Stage 2 column is included in the index - like COL9 above). Data types are required to match until V8.
3. Stage 1 Data Filtering - Any Stage 1 condition that has not been applied in the index entries is applied when the data page is accessed (because all columns live there). Data types are required to match until V8.
4. Stage 2 Data Filtering - Any condition that is not Stage 1 will be applied at this point (an infinite number of possible predicates). This filtering is still better than program filtering which occurs after each element on the result row is transferred to the calling program (one at a time). Any data type mismatches were filtered here until V8.

Visual Plan Graphs - Simple

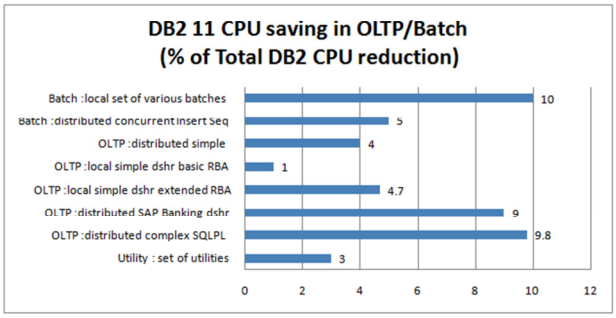


Visual Plan Graphs - Busy



DB2 11 OLTP/Batch Performance Expectations

- These are results from IBM testing
- Performance expectations vary depending on many factors, including
  - Access path selection, Read/Write ratio, Number of rows returned
  - Number and type of columns returned, Number of partitions touched
  - Schema - Number of partitions defined, DPSI, etc
  - RELEASE option, data compression
  - Query costing, query rewrite, index skipping

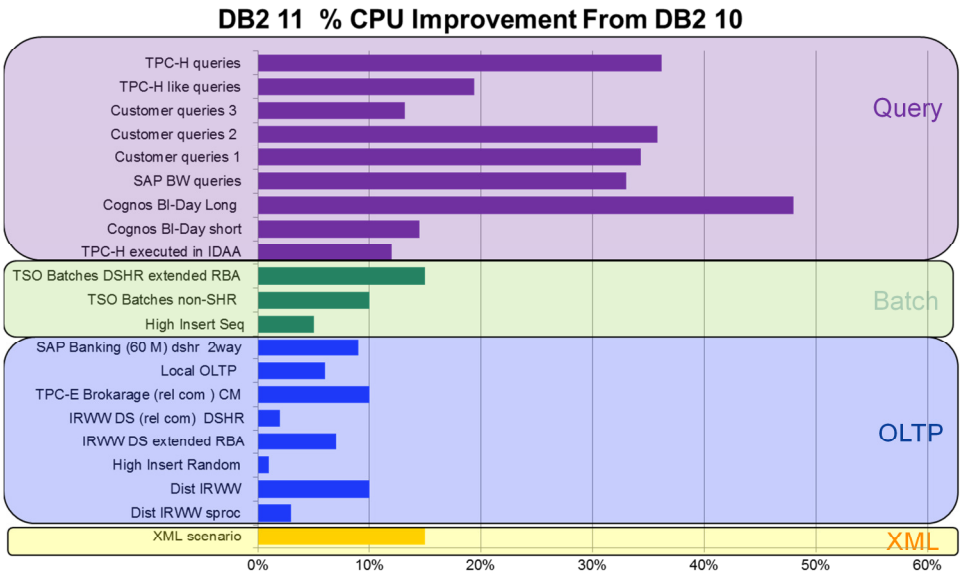


DB2 11 Performance Sweet Spots

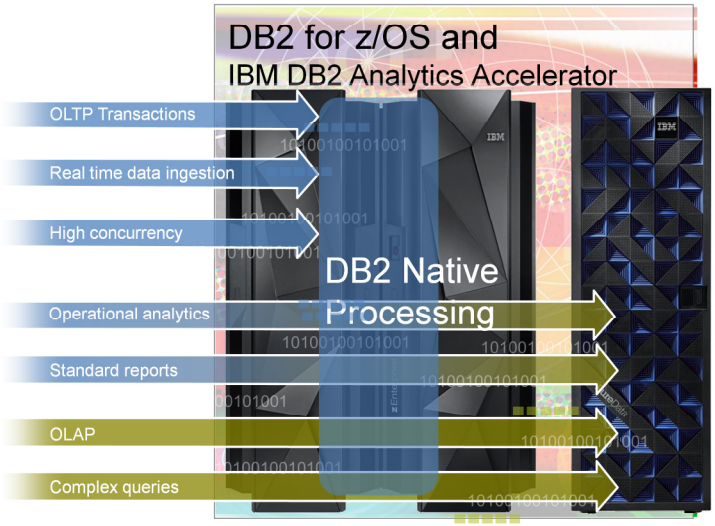
- Write Intensive Batch
- Queries
  - With compressed tables
  - With access path improvement
  - With sort intensive workload
  - Accessing multiple DPSI partitions
  - IDAA with large result sets
- Online transactions
  - Write intensive transactions
  - With large # of partitions (>200 partitions) with REL(COMMIT)
  - With large buffer pools
  - With queries returning a large number of columns
  - Chatty DDF applications with z/OS Communications Server
- Cost saving from zIIP eligible address space SRB time
  - DBM1 in data sharing
  - MSTR address space for update intensive workloads

Just to recap sweet spots in DB2 11,
Queries we have significant access path related improvement
You should see CPU reduction accessing Tables with compressed data
Workload invokes large sorts should see good improvement,

Impressive DB2 11 Performance Results!



DB2 11 The Foundation for Business Critical Analytics



• DB2 11 CPU savings benefit query workloads with or without IBM DB2 Analytics Accelerator

The analytics Accelerator offers significant benefits to address your operational processes.
These include -

- Delivering High-speed analytics that can be easily integrated into operational applications,
- Offering Historical views are quickly analyzed for more train-of-thought analysis
- Enabling decision makers to perform business analysis they never dared in the past
- Providing a secured environment for highly sensitive structured data
- And speeding up batch reporting cycles to meet stricter service level agreements

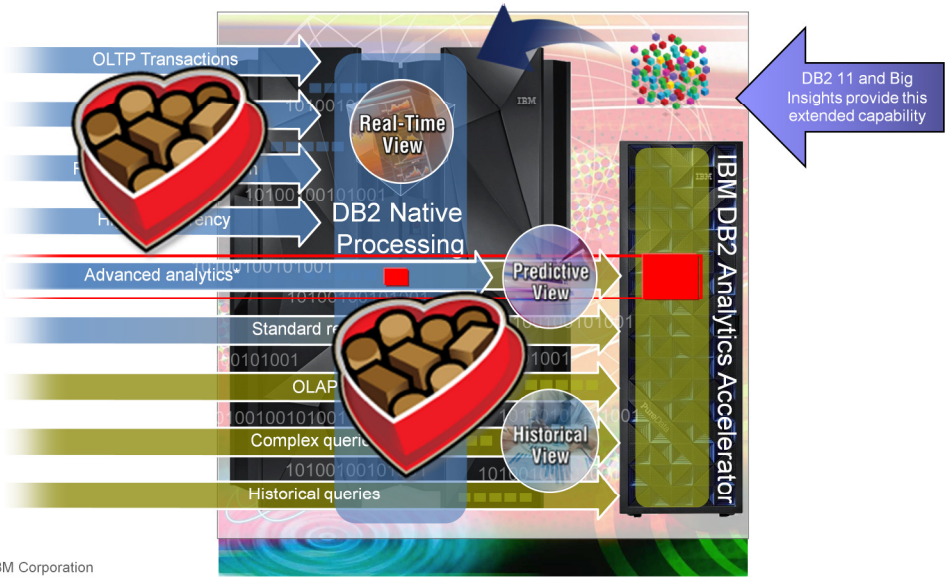
It offers significant cost savings in 3 key areas:

1. Lowers storage costs of housing static, historical data that is often terabytes in size – by moving it from the z/enterprise storage - that is designed for efficiently managing read/write data that is constantly changing.
2. Eliminating the many tuning tasks performed by database administrators on a daily basis such as MQT creation, Index updates etc. created to speed complex queries, it eliminates the hours/days of query tuning by DBAs,
3. while offloading queries to a specially tuned device for processing; enabling organizations to save considerably by reducing the consumption of MIPs within the DB2 LPAR..

An as an appliance the Db2 Analytics Accelerator can be installed in hours and providing value the same day since there is no application changes required. The device is completely transparent to the user, except for the fact that their work is suddenly done dramatically faster.

- **Tightly integrated with DB2 for z/OS**
  - **Transparent to DB2 applications**
  - Inherits DB2 security
  - Enables DB2 to make best choice for query processing
  - Capitalizes on the availability of System z
  - Minimizes data movement
- Leverages Netezza Technology
  - FPGA for hardware query acceleration (These FPGAs are used for data decompression, data filtering and early SQL projections and restriction)
  - **No need for indexes. MQTs, or query plans**
  - Disk mirroring and blade failover
  - Box can be remissioned when new technology is introduced
- Offloads long running queries from System z
- Created for mixed workload of operational transaction systems, data warehouse, operational data stores, and consolidated data marts

DB2/Accelerator Sweet Spots Do Not Overlap



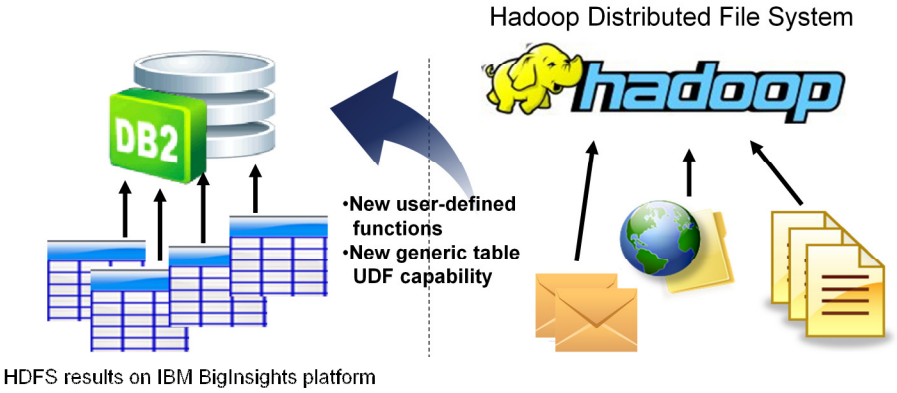
DB2 will always have capabilities to do the best it can with whatever workload you place on it. The DB2 Accelerator will always have the capabilities to the best it can with the workload the DB2 optimizer decides to offload. Each new release of each component will change which queries stay on DB2 and which ones get accelerated.

DB2 optimizer:

- ✓All tables added and enabled
- ✓SQL functionality supported
- ✓Cost and heuristics OK (ENABLE only) for routing

Enhancing DB2 11 Analytics on "z" with Big Data

- Providing the connectors – User Defined Functions
- Providing capability to allow DB2 applications to easily and efficiently access these data sources



Big data and business analytics represent the new IT battleground. Here is some stats:

•IDC estimates the big data market will reach $16.9 billion by 2015, and that enterprises will invest more than $120 billion to capture the business impact of analytics, across hardware, software and services that same year.

•The "digital universe" will grow to 2.7ZB in 2012, up 48% from 2011 and rocketing toward nearly 8ZB by 2015 (IDC).

•53% of business leaders don't have access to the information from across their organizations they need to do their jobs (IBM CMO Study).

•Organizations applying analytics to data for competitive advantage are 2.2x more likely to substantially outperform their industry peers (MIT/IBV Report)

The amount and types of data being captured for business analysis is growing. A classic example of this large superset of data is Web logs, which contain unstructured raw data.

In an increasing trend unstructured data is being stored on new frameworks. These infrastructures encompass hardware and software support such as new file systems, query languages, and appliances. A prime example being Hadoop.

So what is Hadoop?

•A java-based framework that supports data intensive distributed applications and allows applications to work with thousands of nodes and petabytes of data.

•Hadoop framework is ideal for distributed processing of large data sets .

•It utilizes a distributed file system that is designed to be highly fault tolerant and allows high throughput access to data and is suitable for applications that have large data sets.

DB2 11 Support for Big Data Details

- Analytic jobs can be specified using JSON Query Language (Jaql)
  - Submitted to IBM's Hadoop based BigInsights platform
  - Results stored in HDFS format
- A new generic, polymorphic, TABLE User Defined Function (HDFS_READ) reads the Bigdata analytic result from HDFS
  - For subsequent use in an SQL query
- Must have a variable shape of HDFS_READ output table
  - Its output schema is determined at query run-time
  - DB2 11 supports generic table UDFs, enabling this function
- A new function, JAQL_SUBMIT, enables invocation of IBM BigInsights Jaql from a DB2 application
  - Returns the correct URL string for the HDFS result in VARCHAR(512) or ''

The DB2 11 goal is to connect DB2 with IBM's Hadoop based BigInsights big data platform, and to provide customers a way to integrate their traditional applications on DB2 z/OS with Big Data analytics. Analytics jobs can be specified using JSON Query Language (Jaql) and submitted to IBM's Bigdata platform and the results will be stored in Hadoop Distributed File System (HDFS).

DB2 11 plans to integrate DB2 for z/OS with BigInsights from the database side and enable applications on DB2 z/OS to access big data analytics. It will include the ability to submit jobs specified in JSON Query Language (JAQL) to BigInsights and to access the Hadoop file system via user-defined functions.

(Remember that traditional table UDFs require that the output schema of the UDF is specified statically at function creation time. There would be a need to write a different external user-defined table function for reading each different Hadoop files which produce different output schema.

DB2 11 will provide a table UDF (HDFS_READ) to read the Bigdata analytic result from HDFS so that it can used in an SQL query. Since the shape of HDFS_READ's output table varies, we will also support a generic table UDF which improves the usability of HDFS_READ.

There would be a need to write a different external user-defined table function for reading each different Hadoop files which produce different output schema. DB2 11 will implement a new kind user-defined table functions which are called generic table UDFs. Its output schema are determined by at query compile-time. Therefore generic table UDFs are polymorphic, it increases reusability as the same table function can be used read different Hadoop files and produce different output tables.

========

JSON & Jaql

JSON (JavaScript Object Notation), is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for many languages.

The JSON format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application, serving as an alternative to XML.

Jaql is a JSON Query Language, with it's input/output designed for extensibility. Input can be anything that produces json values, and output can be anything that consumes json values. Examples are:

•Any Hadoop InputFormatand OutputFormat

•HDFS files, Facebook'sThrift format, HBasetables, Facebook'sHive partitioned files

•Queries on these are transformed into map/reduce

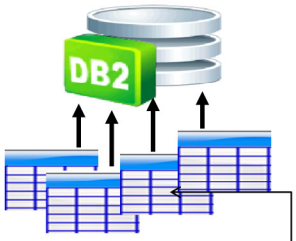
•Local files

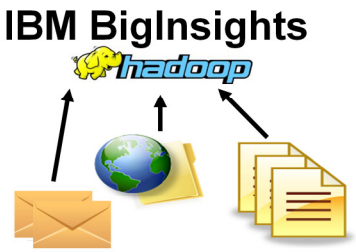
•JDBC sources

Typical Customer Big Data Use Case

4.) DB2 reads the BigInsights result file
SELECT A.* FROM AGENT A JOIN TABLE(HDFS_Read(JAQL_SUBMIT result,")) AS …..

1.) Email from all clients sent to an insurance company ingested into IBM's BigInsights platform





3.) BigInsights creates a file of results (names and email addresses of customers at risk)

2.) DB2 kicks off a Hadoop job on BigInsights - analyzes the emails and identifies customers who have expressed dissatisfaction

DB2 joins the result with the Agent table and sends list of at-risk customers to their mobile device

Hadoop integration for enhanced analytics is a somewhat promising growth driver for z. Our longer term roadmap for BigData integration is to offload to Hadoop, using an IDAA-like approach, which will position us for supporting analytics on XML data and other things and is consistent with our hybrid architecture approach for analytics (using commodity MIPS for cpu intensive analytics work). This is in contrast to LUW's approach of "Common SQL" which is a more tightly integrated approach, which makes more sense on their platform, but it's also a very high development cost.

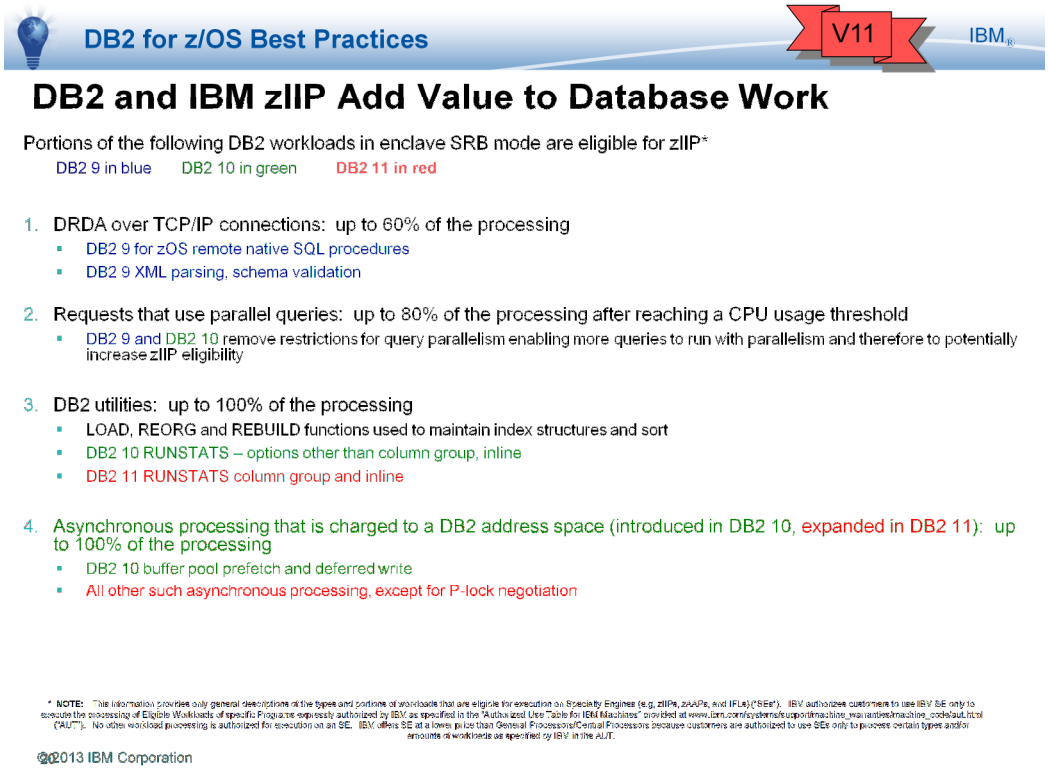
More HDFS_Read Examples

- INSERT INTO T1 SELECT a, b FROM TABLE
  (HDFS_Read('http://9.30.11.27/data/controller/dfs/file1.csv',''))
  AS X(A INT, B DOUBLE, C INT);

- SELECT T1.*, hd.a, hd.b
  FROM T1 JOIN TABLE
  (HDFS_Read('http://9.30.11.27/data/controller/dfs/file1.csv',
  '')) AS X(A INT,B DOUBLE,C INT)
  ON T1.x = X.c;

- CREATE VIEW hadoopData AS
  SELECT * FROM TABLE(
  HDFS_Read('http://9.30.11.27/data/controller/dfs/file1.csv',''))
  AS X(A INT, B DOUBLE, C INT);

  SELECT T1.x, T1.y, hd.a, hd.b
  FROM DB2Data AS T1, hadoopData AS hd
  Where T1.x = hd.c;



Note: IBM establishes the "CPU Usage Threshold" for each System z Machine type. In this example, only such portion of DB2 for z/OS processing is considered an Eligible Workload for the zIIP.

*** zIIP allows a program working with z/OS to have all or a portion of its enclave Service Request Block (SRB) work directed to zIIP. Above types of DB2 work are those running in enclave SRBs, of which portions can be sent to zIIP.**

The zIIP is designed so that a program can work with z/OS to have all or a portion of its enclave Service Request Block (SRB) work directed to the zIIP. The above types of DB2 V8 work are those executing in enclave SRBs, of which portions can be sent to the zIIP. Not all of this work will be run on zIIP. z/OS will direct the work between the general processor and the zIIP. The zIIP is designed so a software program can work with z/OS to dispatch workloads to the zIIP with no anticipated changes to the application – only changes in z/OS and DB2.

IBM DB2 for z/OS version 8 was the first IBM software able to take advantage of the zIIP. Initially, the following workloads can benefit:

• SQL processing of DRDA network-connected applications over TCP/IP: These DRDA applications include ERP (e.g. SAP), CRM (Siebel), or business intelligence and are expected to provide the primary benefit to customers. Stored procedures and UDFs run under TCBs, so they are not generally eligible, except for the call, commit and result set processing. DB2 9 remote native SQL Procedure Language is eligible for zIIP processing. BI application query processing utilizing DB2 parallel query capabilities; and functions of specified DB2 utilities that perform index maintenance.

•For more, see http://www.ibm.com/systems/z/ziip/

2010 New method to control the portion of SQL requests that are authorized to be diverted to zIIP engines with improved performance via reduced processor switching. This change also increases portion of DRDA that is authorized to run on zIIPs to 60%. APAR PM12256 for V8 & DB2 9. Included in DB2 10 base. PM28626 to resolve some anomalies.

DB2 10 improvements include increased parallel processing, the RUNSTATS utility and buffer pool prefetch.

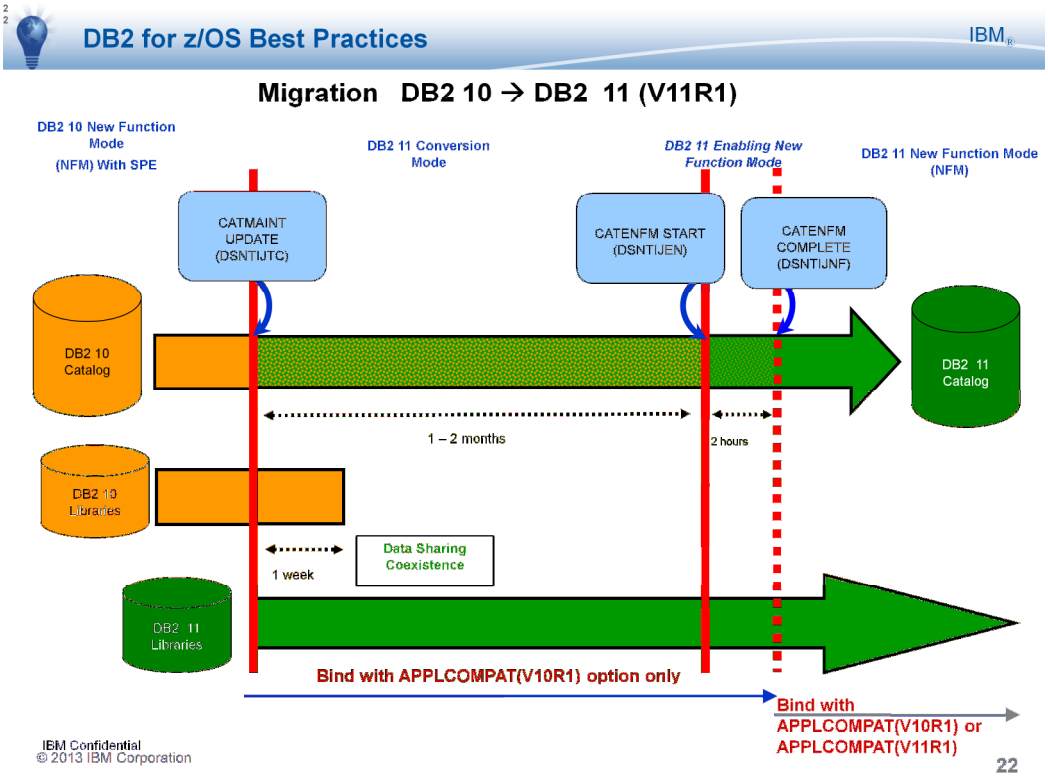
V11: all SRB mode unrelated system agents, except P-lock negotation (response time critical). 1) the processing of pseudo deleted index entries, 2) XML multi-version documents cleanup (available in DB2 10 for z/OS via APAR PM 72526), and 3) such processing that executes within the MSTR address space, such as log write and log read.

Easier DB2 Version Upgrade – application compatibility

- New DB2 releases can introduce SQL behavior changes which can break existing applications
  - For example, changes for SQL standards compliance
  - Example: DB2 10 CHAR function with decimal input no longer returns leading zeros when there is a decimal point
- Application Compatibility (APPLCOMPAT) – new option for enforcement
  - **APPLCOMPAT(WARN)**
  - Provide mechanism to identify applications affected by SQL changes
  - Provide seamless mechanism to make changes at an application (package) level or at a system level
  - **APPLCOMPAT(VnnR1)** – nn is the DB2 Version Number. V10R1 is the lowest release of DB2 catered for
  - **CURRENT APPLICATION COMPATIBILITY –** Dynamic SQL special register
    - This mechanism will enable support for up to two back level releases (N-2)
    - The release after DB2 10 will be the initial deployment of this capability
    - DB2 10 will be the lowest level of compatibility supported

V10: ZParm BIF_COMPATIBILITY to revert to the way it worked in V9. Can also set SYSCOMPAT_V9 to beginning of PATH Bind option or in CURRENT PATH. IFCID 366 Introduced to report when CHAR with decimal used

N8195 – SQL Compatibility. ZPARM for Default BIND Option. BIND/REBIND options for Packages. Special Register for Dynamic SQL (CURRENT APPLICATION COMPATIBILITY). Warnings provided when program uses incompatible SQL. IFCID 366 will report on Packages affected in both modes and Dynamic SQL. IFCID 376 is a roll up of 366, one record written for each unique static or dynamic statement. Migration job DSNTIJPM will also warn of static SQL packages affected before SQL used. APPLCOMPAT(VnnR1) – nn is the DB2 Version Number. V10R1 is the lowest release of DB2 catered for. V10R1 Only Allowed in CM. V11R1 or V10R1 Allowed in NFM. APPLCOMPAT(V10R1) assumed for all static SQL packages bound prior to V10



This diagram shows the migration process, what is highlighted at the bottom of the slide is that if any BINDS or REBINDS take place in Conversion mode then they can only use APPLCOMPAT(V10R1) as no new SQL functions for DB2 11 can be used in conversion mode.

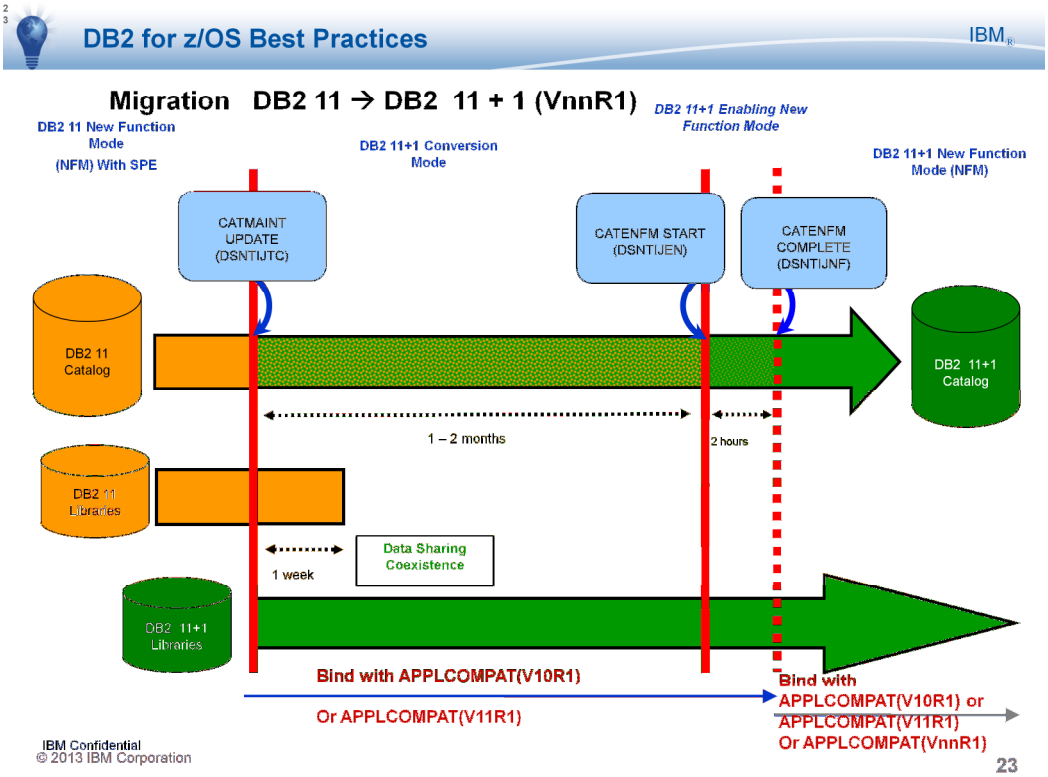
APPLCOMPAT(V10R1) is assumed in conversion mode if not specified.

But when in new function APPLCOMPAT(V10R1) can be used to allow SQL that valid in DB2 10 but is not valid in DB2 11 or APPLCOMPAT(V11R1) so that new SQL functionality can be used.

APPLCOMPAT(V10R1) on BIND REPLACE in V11 NFM stops use of new SQL functionality introduced in V11 NFM.

Use of APPLCOMPAT(V11R1) is required in DB2 11 NFM on BIND ADD|REPLACE in order to use new SQL functionality as introduced in 11 NFM.

22



Vss is the version number for DB2 11 and VnnR1 is the version number for the next version of DB2 after DB2 11.

This diagram shows the migration process, what is highlighted at the bottom of the slide is that if any BINDS or REBINDS take place in Conversion mode then they can use APPLCOMPAT(V10) or APPLCOMPAT(V11R1) as no new SQL functions for DB2 11 can be used in conversion mode.

APPLCOMPAT(V10) will allow SQL that may not be valid in DB2 11 or in the version of DB2 after DB2 11 but was valid in DB2 10.

But when in new function APPLCOMPAT(V10R1), APPLCOMPAT(V11R1) or APPLCOMPAT(VnnR1) can be used

APPLCOMPAT(V10R1) - to allow SQL that valid in V10 but is not valid in DB2 11 or DB2 Sequioa + 1.

APPLCOMPAT(V11R1) - to allow SQL that was valid in DB2 11 but not in DB2 11 + 1.

APPLCOMPAT(VnnR1) - to allow SQL that is only valid in DB2 Sequia + 1.

23

Easier DB2 Version Upgrade...

- Faster ENFM processing
  - Lab measurement showed 16x faster in V11 vs. V10 using a large customer catalog
  - Note: V11 ENFM performance is sensitive to size of SYSLGRNX. Consider running MODIFY RECOVER to clean up old entries if SYSLGRNX is very large
- Access path stability improvements
- Higher code quality stability levels
- New workload Capture/Replay tooling can help testing of DB2 version upgrades

Based on Mai's measurement against Hewitt catalog, DSNTIJEN shows 18x elapsed time improvement (1660 sec vs. 92 sec) with DB2 11 compared to DB2 10.

Performance Improvements – no REBIND needed

- DDF performance improvements
  - Reduced SRB scheduling on tcp/ip receive using new CommServer capabilities
  - Improved autocommit OLTP performance
  - DRDA package based continuous block fetch
- xProcs above the bar
  - 31-bit Vstor relief enabled by RMODE 64 support in z/OS 1.13 and above
  - Enables other internal performance improvements
- zIIP enablement for all SRB-mode DB2 system agents that are not response time critical
- Avoid cross-memory overhead for writing log records
- INSERT performance
  - Latch contention reduction for classes 6, 14, 19
  - CPU reduction for Insert column processing and log record creation
  - Data sharing LRSN spin avoidance
  - Page fix/free avoidance in GBP write

Suppress-null indexes: EXCLUDE NULL KEYS on CREATE INDEX. Will prevent Index entries being created when **ALL** values for indexed columns are actually NULL

N244 DDF performance.

N2645 xProcs above the bar. z./OS 1.13 and above supports RMODE 64. Moving more control data into the SPAB (DN1607) to reduce the cost of moving a section from the EDM Pool (or Global Statement Cache) to thread storage can only be done if the xPROCs can be moved ATB.

N2895 DSCF performance.

N4504 General Performance improvements:

260 Reduce section storage movement (DN1607).

N265 – Insert space search improvements.

Insert: DB2 V10 has had a significant Insert performance enhancement, both in terms of CPU reduction and elapsed time, ie scalability enhancement, in general. DB2 V11 continues the same trend with contention and/or wait time reduction for latch class 6, 14, 19, and CPU reduction for Insert column processing, log record creation, page fix/free avoidance in GBP write, etc.

DSCF/IFC enhancements: Move WWFR, CCB above the bar

Performance Improvements – no REBIND needed…

- Automatic index pseudo delete cleanup
  - DBA work would be required for fine tuning
- ODBC/JDBC type2 performance improvements
  - Stored procedure invocation
- Java stored procedure multi-threading improvements
- Sort performance improvements
- DPSI performance improvements for merge
- Performance improvements with large number of partitions
- XML performance improvements
- Optimize RELEASE(DEALLOCATE) execution so that it is consistently better performing than RELEASE(COMMIT)
  - Monitor # parent locks and cleanup internal structures when threshold is hit
- IFI 306 filtering capabilities to improve Replication capture performance
- Utilities performance improvements

Suppress-null indexes: EXCLUDE NULL KEYS on CREATE INDEX. Will prevent Index entries being created when **ALL** values for indexed columns are actually NULL

N244 DDF performance.

N2645 xProcs above the bar. z./OS 1.13 and above supports RMODE 64. Moving more control data into the SPAB (DN1607) to reduce the cost of moving a section from the EDM Pool (or Global Statement Cache) to thread storage can only be done if the xPROCs can be moved ATB.

N2895 DSCF performance.

N4504 General Performance improvements:

260 Reduce section storage movement (DN1607).

N265 – Insert space search improvements.

Insert: DB2 V10 has had a significant Insert performance enhancement, both in terms of CPU reduction and elapsed time, ie scalability enhancement, in general. DB2 V11 continues the same trend with contention and/or wait time reduction for latch class 6, 14, 19, and CPU reduction for Insert column processing, log record creation, page fix/free avoidance in GBP write, etc.

DSCF/IFC enhancements: Move WWFR, CCB above the bar

Performance Improvements – no REBIND needed

- ACCESS DATABASE command performance
- DGTT performance improvements
  - Avoid incremental binds for reduced cpu overhead
- P-procs for LIKE predicates against Unicode tables
- Improved performance for ROLLBACK TO SAVEPOINT
- zEC12 integration for performance improvements
  - Pageable 1M frames for DB2 CPU savings (requires Flash Express)
    - Buffer pool control structures (retrofit to V10)
    - DB2 executable code (requires z/OS 2.1)
  - 2G page frame size to position for extremely large main memory sizes
  - Optimizer CPU and I/O cost balancing improvements (can also benefit z196 and z10)
- Latch contention reduction and other high n-way scalability improvements
- Data sharing performance improvements
- LRSN spin reduction with extended LRSNs
- Castout performance
- GBP write-around
- Index split performance

N269 Castout Improvements. Overlapping RFCO for pageset castout. Avoid sending page lists for GBP structure threshold and GBP checkpoint.

N207 – reduce data sharing index log force

ACCESS DATABASE: S10724 ACCESS DB runs under a separate service task so it doesn't cause queueing for other DB commands. S2927 ACCESS DB command to run in parallel

N254 Temp table performance/usability: not logged DGTTs – new NOT LOGGED option for DGTTs, default is LOGGED. Keep static SQL statements that use DGTTs prepared across commit; Explain on incrementally bound queries; Additional stats collection on temp tables.

Rollback to savepoint: in prior releases, performance degradation occurred due to increasing # of log records being scanned with each rollback request. In V11 we remember the point in the log where the previous rollback completed.

1M for DB2 code. Reduces TLB misses. 1.8% improvement for IRWW. Requires z/OS 2.1 or above.

Removed the latch contention for PB directory access (apar is opened to retrofit to V10)

Removed the latch contention for fast path get page (used by index mgr) to locate a root page

Performance Improvements – REBIND required (with or without APREUSE)

- Query transformation improvements – less expertise required to write performance SQL
  - Enhanced query rewrite to improve predicate indexability
    - new situations where non-indexable predicates can be rewritten by Optimizer to be indexable
    - Convert some common stage 2 predicates to indexable (YEAR(), DATE(), SUBSTR(col,1,x), value BETWEEN COL1 AND COL2)
    - Improved indexability for OR COL IS NULL predicates
    - Push complex predicates inside materialized views/table expressions
  - Enhanced pruning of "always true" and "always false" predicates
- Enhanced duplicate removal
  - Lots of queries require duplicate removal: e.g. DISTINCT, GROUP BY, etc.
  - Dup elimination via sorting can be expensive
  - New techniques: Index duplicate removal, early out
  - Will not show in Explain table, need to look at IXSCAN_SKIP_DUPS column in DSN_DETCOST_TABLE to determine if sort avoided
- DDF and RDS runtime result set optimizations
  - Reduced DB2 CPU for IDAA queries

Story 10699 – convert correlated subquery to non-correlated to exploit index access for Union/Union All (e.g. temporal) and some others

N2366 Index skipping: skipping over duplicates when an index provides order for DISTINCT or GROUP BY.

Expression evaluation: Optimized CASE expression evaluation, Sharing of repeat expressions (from view merge), Avoid repeat evaluation of non-column expressions

Rewritten version of predicate written to DSN_PREDICATE_TABLE (original predicate used if internally rewritten predicate is not valid syntax)

For TPC-H we saw 12% CPU reduction for IDAA.

Performance Improvements – REBIND required (with or without APREUSE)…

- In-memory techniques
  - In-memory, reusable workfile
  - Sparse index (limited hash join support)
  - Non-correlated subquery using MXDTCACH
  - Correlated subquery caching
- Non correlated subquery with mismatched length
- Data decompression performance improvement
- Select list do-once
  - Non column expressions in the select list can be executed once rather than per-row
- Column processing improvements
  - Xproc (generated machine code) for output column processing
  - Optimized machine instructions for input/output column processing

“Limitied hash join”: In a nutshell, our hash join implementation works well for joins to small tables - especially when very large tables are joined to small tables. It equally applies to non-correlated subqueries (extended memory usage in V11 - as pre-V11 we were always limited to sparse index of 32K), and small code tables, as well as DW style dimension tables.

"SELECT list do-once" refers to non-column expressions in the select list. Pre-V11 we executed these once per row, and now we execute them once. For example:

SELECT C1, CURRENT DATE - 1 MONTH FROM T1;

In the above example "CURRENT DATE - MONTH" will be evaluated once for the query, rather than once per qualified row (as in V10).

N195: Column processing improvements: xproc for column procedure for output column, it is applicable for distributed output column (select). Also code change on MVCDK for local input/output column processing

Performance Improvements – REBIND required (with or without APREUSE)…

- RID overflow to workfile handled for Data Manager set functions
  - DB2 10 added RID overflow to workfile
  - DB2 11 adds support for set functions (COUNT, MAX, MIN etc) which was excluded in DB2 10
- Performance improvements for common operators
  - MOVE, CAST, output hostvar processing, CASE, SUBSTR, DATE, others
- DECFLOAT data type performance improvements
  - Up to 23% CPU reduction for conversion to/from decfloat
  - Approx. 50% cpu reduction in INSERT, FETCH for decfloat columns
  - Helped further by zEC12 hw improvements for decimal floating point

Performance Improvements – REBIND required (without APREUSE)

- DPSI and page range performance improvements
  - Page range screening for join/correlation predicates
  - Parallelism optimization for DPSI access
- Optimizer CPU and I/O cost balancing improvements
  - Measured results: 3% to >30% performance improvement for query workloads

Performance Improvements – DBA or application effort required

- Suppress-null indexes
  - Index entries not created when all values for indexed columns are NULL
  - Reduced index size, improved insert/update/delete performance, compatibility with other DBMSes
  - Improved utility CREATE INDEX performance
- New PCTFREE FOR UPDATE attribute to reduce indirect references
- DGTT performance improvements
  - Non logged DGTTs
- Global variables
  - Easier, more efficient sharing of data between SQL statements

Suppress-null indexes: EXCLUDE NULL KEYS on CREATE INDEX. Will prevent Index entries being created when **ALL** values for indexed columns are actually NULL

N244 DDF performance.

N2645 xProcs above the bar. z./OS 1.13 and above supports RMODE 64. Moving more control data into the SPAB (DN1607) to reduce the cost of moving a section from the EDM Pool (or Global Statement Cache) to thread storage can only be done if the xPROCs can be moved ATB.

N2895 DSCF performance.

N4504 General Performance improvements:

260 Reduce section storage movement (DN1607).

N265 – Insert space search improvements.

Insert: DB2 V10 has had a significant Insert performance enhancement, both in terms of CPU reduction and elapsed time, ie scalability enhancement, in general. DB2 V11 continues the same trend with contention and/or wait time reduction for latch class 6, 14, 19, and CPU reduction for Insert column processing, log record creation, page fix/free avoidance in GBP write, etc.

DSCF/IFC enhancements: Move WWFR, CCB above the bar

Performance Improvements – DBA or application effort required

- Extended optimization - selectivity overrides (filter factor hints)
  - Improve optimizer's ability to find the cheapest access path
  - Collect filter factors for predicates in a Selectivity Profile
  - Selectivity Profile is populated via BIND QUERY
- Open dataset limit raised to 200K
- Optimizer externalization of missing/conflicting statistics
  - Identify missing statistics during bind/prepare/explain
  - DBA or tooling to convert output to RUNSTATS input

Suppress-null indexes: EXCLUDE NULL KEYS on CREATE INDEX. Will prevent Index entries being created when **ALL** values for indexed columns are actually NULL

N244 DDF performance.

N2645 xProcs above the bar. z./OS 1.13 and above supports RMODE 64. Moving more control data into the SPAB (DN1607) to reduce the cost of moving a section from the EDM Pool (or Global Statement Cache) to thread storage can only be done if the xPROCs can be moved ATB.

N2895 DSCF performance.

N4504 General Performance improvements:

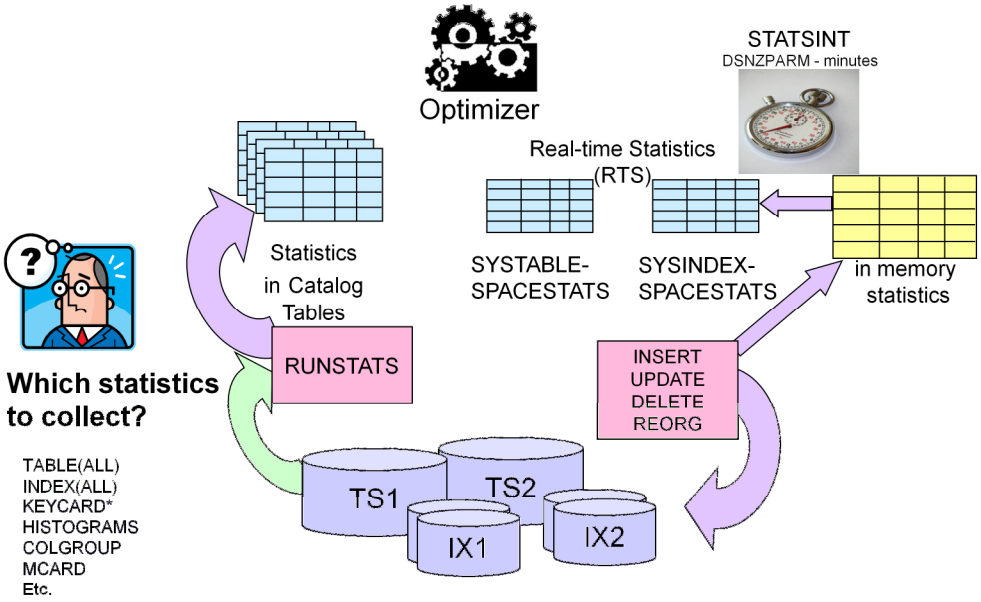
260 Reduce section storage movement (DN1607).

N265 – Insert space search improvements.

Insert: DB2 V10 has had a significant Insert performance enhancement, both in terms of CPU reduction and elapsed time, ie scalability enhancement, in general. DB2 V11 continues the same trend with contention and/or wait time reduction for latch class 6, 14, 19, and CPU reduction for Insert column processing, log record creation, page fix/free avoidance in GBP write, etc.

DSCF/IFC enhancements: Move WWFR, CCB above the bar

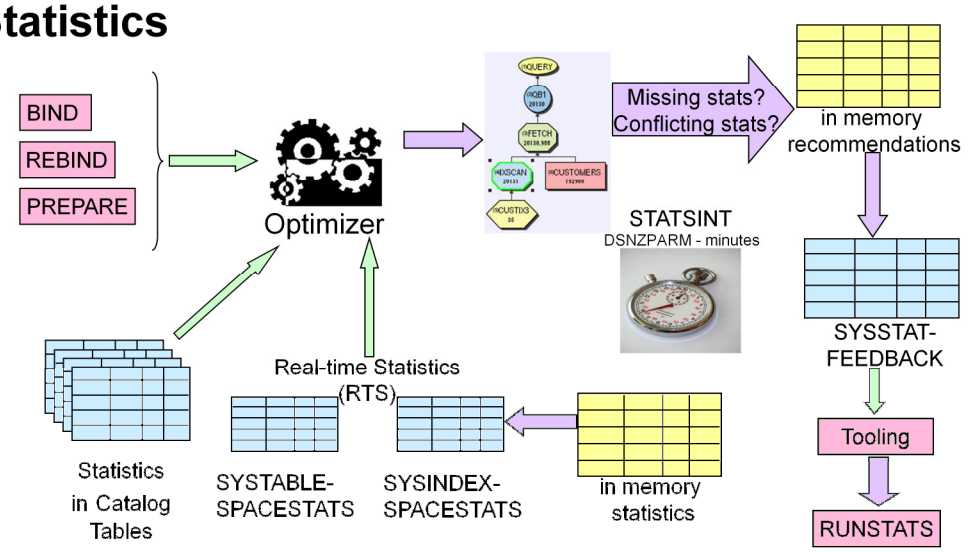
Current Process for Populating Statistics



Focus on where the statistics come from that the optimizer uses.

1. DBA's (or automated tools) run the RUNSTATS utility, which samples the table spaces and indexes and populates various catalog tables with statistics about the number of rows, columns, data values, index keys, cardinality, etc, etc. RUNSTATS can be expensive, depending on options taken and variety of statistics collected. One of the challenges (below) is what statistics to collect, but also challenge – how often to run RUNSTATS to reflect significant changes but not consume too much CPU / elapsed time on RUNSTATS itself.
2. Various processes that change tables and indexes (Insert, Update, Delete) plus REORGs, LOADS, etc, change the tables and information about these processes is stored in DB2 memory, then asynchronously DB2 populates the RTS tables. Async based on DSNZPARM value of STATSINT (minutes)
3. Again, DBA wants to balance collecting sufficient statistics for optimizer to make good choices with cost of executing RUNSTATS. DBA is not always aware how skewed the data values are in the various tables and indexes. So may not make the best choice. That is the problem this epic addresses
4. Asterisk (*) by KEYCARD because starting in DB2 10 KEYCARD is automatic and cannot be removed.

DB2 11 – Optimizer Externalization of Missing Statistics



Solution: For any execution of BIND, REBIND or PREPARE, the optimizer will make note of which missing or conflicting statistics may have helped determine an appropriate access path. DB2 will keep these recommendations within memory and asynchronously populate the new SYSIBM.SYSSTATFEEDBACK table. That table, with appropriate tooling to interpret the recommendations, can be used to provide RUNSTATS with the right options for the corresponding statistics the next time RUNSTATS runs. During access path determination, optimizer knows when assumptions/estimates are taken to compensate for missing statistics. Goal is to externalize this information as input for Runstats, so that runstats automatically collects the missing statistics at next execution. Ideally, we should not need stats adviser any longer. As a result, we expect significant CPU reduction because of better access path selection as well as fewer access path selection related pmrs.

ZPARM STATSINT is reused as the async timer for externalizing these recommendations (as well as externalizing RTS stats). New zparm STATFDBK_SCOPE. STATFDBK_SCOPE can be set to allow recommendations for all queries, only static queries, only dynamic queries. It can also be used to disable statistics recommendations. Default is all SQL.

A new column in SYSTABLES named STATS_FEEDBACK provides control of statistics recommendations at the table level.

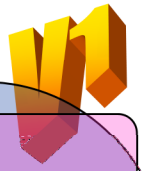
Note: RUNSTATS will delete the recommendation from SYSSTATFEEDBACK when it uses it. So SYSSTATFEEDBACK cannot be used to track history, and while SYSSTATFEEDBACK can be quite large, it is shown here smaller than DSN_STAT_FEEDBACK on next chart because once a recommendation is used by RUNSTATS, that recommendation row is removed from the table.

2 categories of concerns for access path costing and statistics. 1: skewed data 2. everything else. For 2. everything collected is useful, regardless if whether using host variables. For 1. only valuable if know the value of the host variable, otherwise optimizer does not get any help.

Issue: optimizer finds stats most useful when the predicate is evaluating a literal. So consider when to use literals: when to use host variables or parameter markers (when things change frequently) vs. literals, when the value never changes. Note: many SQL statements use host variables throughout, even for predicates that do not change from one execution to the next. These queries can be costed more effectively if the unchanging predicates are coded as literals, even for static SQL, because the optimizer will see the statistics for the value that is actually used, rather than coming up with a 'guess' of what the host variable will be when the statement executes.

For dynamic SQL, REOPT(AUTO) can get around the problem of optimizer not knowing the values, because it will re-evaluate. May reduce flushing the cache, since optimizer won't re-prepare every time, only when filter factors have substantially changed. Then only helpful if you have the skewed statistics.

IBM SQL History - DB2 V1 vs. DB2 9 SQL



Inner Joins, Subqueries, GROUP BY, HAVING, ORDER BY, 21+ Built-in Functions

Table Expressions, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions, Limited Fetch, Scrollable Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Table Support, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, IS NOT DISTINCT FROM, ON COMMIT DROP, Transparent ROWID Column, GET DIAGNOSTICS, Stage1 unlike data types, Multi-row INSERT, Multi-row FETCH, Dynamic Scrollable Cursors, Multiple CCSIDs per statement, Enhanced UNICODE, and Parallel Sort, TRUNCATE, DECIMAL FLOAT, VARBINARY, optimistic locking, FETCH CONTINUE, MERGE, call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, client special registers, long SQL Object names, SELECT FROM INSERT, UPDATE, DELETE, MERGE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY IN subselect and fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables

DB2's SQL had humble beginnings.

Stage1 unlike data types, Multi-row INSERT, FETCH, Multi-row cursor UPDATE, Dynamic Scrollable Cursors, Multiple CCSIDs per statement, GET DIAGNOSTICS, Enhanced UNICODE, IS NOT DISTINCT FROM, VARBINARY, FETCH CONTINUE, MERGE

Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scrollable Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Table Support, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, client special registers, long SQL Object names, SELECT FROM INSERT, UPDATE, DELETE, MERGE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY IN subselect and fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables, DECIMAL FLOAT, XQuery,TRUNCATE, OLAP Functions, Session variables, OmniFind, Spatial, ROLE

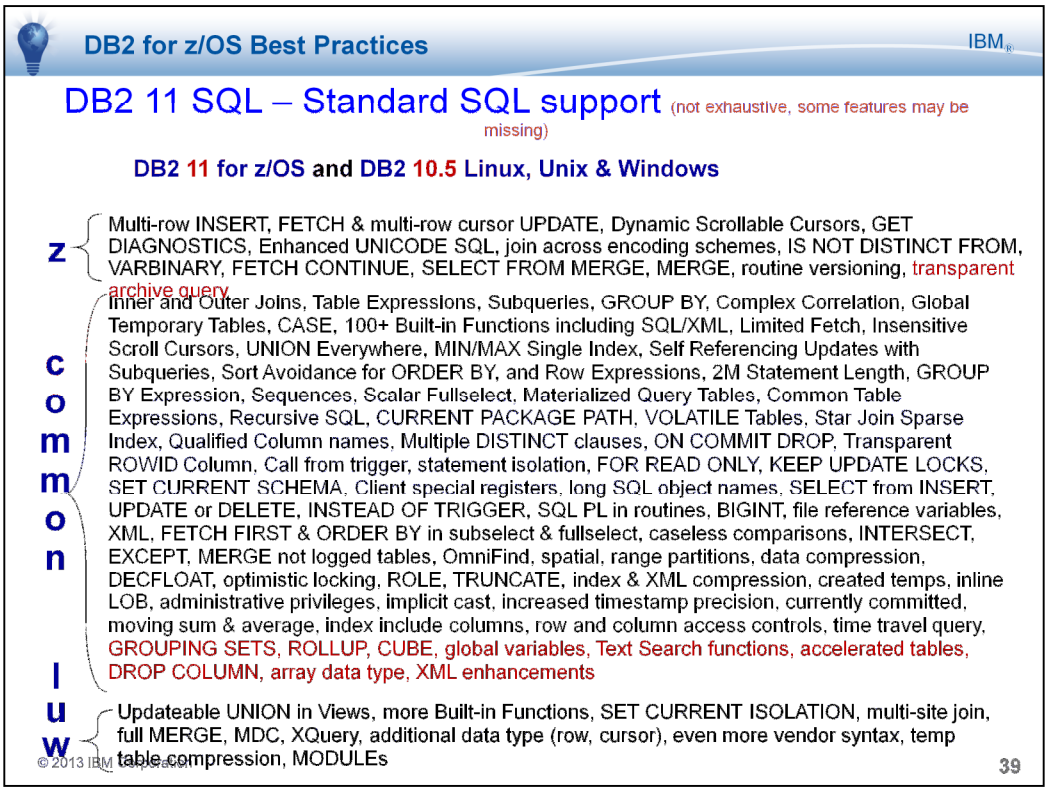
GROUPING SETS, ROLLUP, CUBE, Many Built-in Functions, SET CURRENT ISOLATION , multi-site join, MERGE, ARRAY data type, global variables, Oracle syntax, XML enhancements

DB2 10 z/OS vs. DB2 10 LUW

Multi-row INSERT, FETCH & multi-row cursor UPDATE, Dynamic Scrollable Cursors, GET DIAGNOSTICS, Enhanced UNICODE SQL, join across encoding schemes, IS NOT DISTINCT FROM, VARBINARY, FETCH CONTINUE, MERGE, SELECT from MERGE, routine versioning, timestamps w/timezone

Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including SQL/XML, Limited Fetch, Insensitive Scroll Cursors, UNION Everywhere, MIN/MAX Single Index, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions, 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Tables, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column, Call from trigger, statement isolation, FOR READ ONLY KEEP UPDATE LOCKS, SET CURRENT SCHEMA, Client special registers, long SQL object names, SELECT from INSERT, UPDATE or DELETE, INSTEAD OF TRIGGER, Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH FIRST & ORDER BY in subselect & fullselect, caseless comparisons, INTERSECT, EXCEPT, not logged tables, OmniFind, spatial, range partitions, data compression, session variables, DECIMAL FLOAT, optimistic locking, ROLE, TRUNCATE, index & XML compression, created temps, inline LOB, administrative privileges, implicit cast, date/time changes, currently committed, moving sum & average, index include columns, row and column access control, time travel query, XML enhancements

Updateable UNION in Views, GROUPING SETS, ROLLUP, CUBE, more Built-in Functions, SET CURRENT ISOLATION, multi-site join, MERGE, MDC, XQuery,, additional data type (array, row, cursor), global variables, even more vendor syntax, temp table compression, MODULEs



This chart shows the relationship of DB2 for Linux, Unix & Windows with
There are three sets of SQL noted above, with some that is unique to DI
The Cross-Platform SQL Reference Version 4.1 documents the prior co
Cross-Platform Development Version 4.1, http://www.ibm.com/developerworks/db2/library/techarti

Expanded Analytics Capabilities...

- SQL Grouping Sets, including Rollup, Cube
  - Rollup is helpful in providing subtotaling along a hierarchical dimension such as time or geography
  - CUBE is helpful in queries that aggregate based on columns from multiple dimensions
- DB2 support for IDAA V3 and V4 (rolled back to V10)
  - Support for static SQL
  - Propagating DB2 changes to the accelerator as they happen – V11 improved CDC capture performance with new IFI 306 filtering capabilities
  - Detect staleness of data via RTS
  - Reducing disk storage cost by archiving data in the accelerator and maintaining the excellent performance for analytical queries: *High Performance Storage Saver*
  - Workload Manager integration and better monitoring capabilities
  - Increasing the query off-load scope via new special register CURRENT QUERY ACCELERATION
- High performance SPSS in-database scoring via PACK/UNPACK (rolled back to v10)
- Hadoop access via table UDF
  - UDFs shipped with BigInsights
  - Uses new V11 generic table UDF capability
- JSON support

Hadoop access: REST API. BigInsights Fixpacks scheduled for May and next release is Oct. of 2013.

N139 Grouping Sets. Look at last line of example. In it are 3 groups, one result set for each of the 3 with one pass of the data.. Has been supported in LUW for a while. For more sophisticated groupings. Multiple GROUP BY in a single SELECT, for example:

```
SELECT T.YEAR, C. RETAILER, P.LINE,
     SUM(F.UNITSSOLD) AS UNITSOLD
FROM FACTVARS F, TIMELEVEL T, CUSTLEVEL C,
          PRODLEVEL P, CHENLEVEL CH
WHERE F.TIME_LEVEL = T.MONTH AND
    F.CUSTOMER = C.STORE AND
    F.PRODUCT = P.CODE AND
    F.CHANNEL = CH.BASE AND
    F.CHANNEL in ('G9GUPNDE0714')
GROUP BY GROUPING SETS(T.YEAR, C.RETAILER, P.LINE);
```

SQL Rollup, Cube Avoids convoluted SQL for this capability

Rollup is very helpful for subtotaling along a hierarchical dimension such as time or geography

```
ROLLUP(Year, Month, Day) =>
GROUPING SETS((Year, Month, Day),
        (Year, Month),
        (Year),
        ())
```

CUBE is helpful in queries that aggregate based on columns from multiple dimensions.

```
CUBE(YEAR, store, Line) =>
        GROUPING SETS((YEAR, store, Line),
                (Year, Store),
                (Year, Line),
                (Store, Line),
                (Year),
                (Store),
                (Line),
        () )
```

GROUP BY GROUPING SETS

- A grouping sets specification allows multiple grouping clauses to be specified in a single statement.
- DB2 11 introduces support for 3 additional variants of GROUP BY
  - GROUP BY GROUPING SET: fundamental building block for GROUP BY operations
  - GROUP BY ROLLUP: Produces sub-total rows in addition to regular grouped rows.
  - GROUP BY CUBE: Produces row summaries and grand totals

- ***Example:*** Create a result set from the SALES table based on person and date.

```
SELECT WEEK(SALES_DATE) as WEEK, DAYOFWEEK(SALES_DATE) AS DAY,
    SALES_PERSON, SUM(SALES) AS SOLD
FROM SALES
WHERE SALES_DATE > '1999-12-31'
GROUP BY GROUPING SETS
        (WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON)
```


A grouping sets specification allows multiple grouping clauses to be specified in a single statement. This can be thought of as a union of 2 or more groups of rows into a single result set in a single pass. Previously, you would have needed to run multiple queries to achieve the same result. It is logically equivalent to the union of multiple subselects with the group by clause in each subselect corresponding to one grouping set.

Note: grouping sets are the fundamental building blocks for GROUP BY operations. A simple GROUP BY with a single column can be considered a grouping set with one element.

GROUP BY GROUPING SETS results

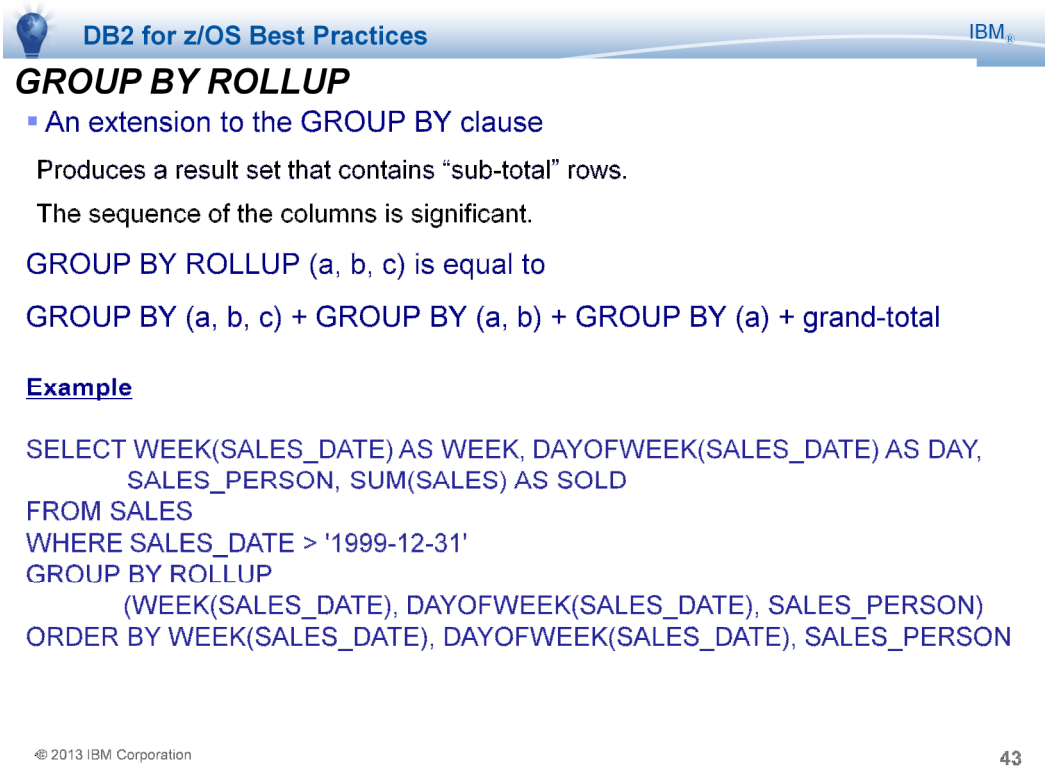
	WEEK	DAY	SALES_PERSON	SOLD
1	NULL	NULL	GOUNOT	50
2	NULL	NULL	LEE	89
3	NULL	NULL	LUCCHESSI	14
4	NULL	4	NULL	25
5	NULL	5	NULL	45
6	NULL	6	NULL	32
7	NULL	7	NULL	51
8	13	NULL	NULL	145
9	53	NULL	NULL	8

An example of the grouping of data is shown by the braces. Therefore we have a group for

1.Sales person with totals for each sales person (RED Braces)

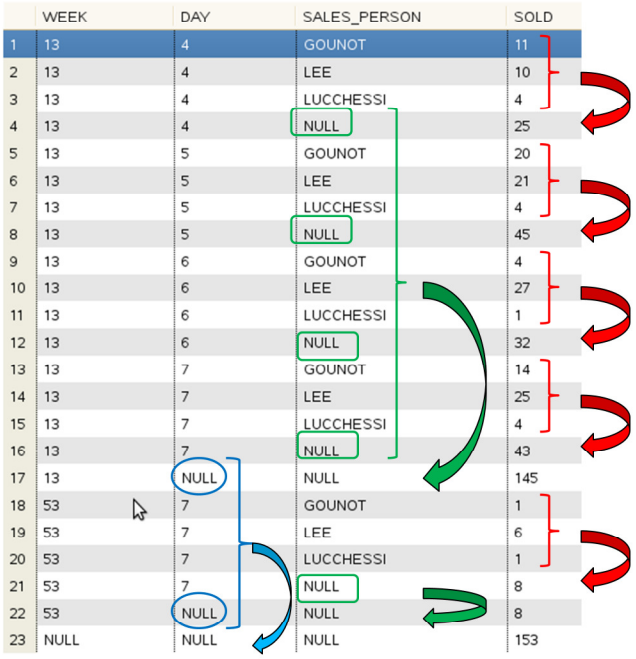
2.Day of Week with total sales for each day (GREEN Braces)

3.Week number with total sales for each week (BLUE Braces)



A ROLLUP group is an extension to the GROUP BY clause that produces a result set that contains *sub-total* rows
In the testing this has resulted significant savings and the savings appear to increase as complexity and number o

GROUP BY ROLLUP results



	WEEK	DAY	SALES_PERSON	SOLD
1	13	4	GOUNOT	11
2	13	4	LEE	10
3	13	4	LUCCHESSI	4
4	13	4	NULL	25
5	13	5	GOUNOT	20
6	13	5	LEE	21
7	13	5	LUCCHESSI	4
8	13	5	NULL	45
9	13	6	GOUNOT	4
10	13	6	LEE	27
11	13	6	LUCCHESSI	1
12	13	6	NULL	32
13	13	7	GOUNOT	14
14	13	7	LEE	25
15	13	7	LUCCHESSI	4
16	13	7	NULL	43
17	13	NULL	NULL	145
18	53	7	GOUNOT	1
19	53	7	LEE	6
20	53	7	LUCCHESSI	1
21	53	7	NULL	8
22	53	NULL	NULL	8
23	NULL	NULL	NULL	153

The key to translating the format of the query results is to recognize that the output format is dependent on the ORDER BY statement. In the example the output is sequenced first on the WEEK, then DAY and finally by SALES PERSON, with a summary or ROLLUP row inserted based on the order of the ROLLUP statement.

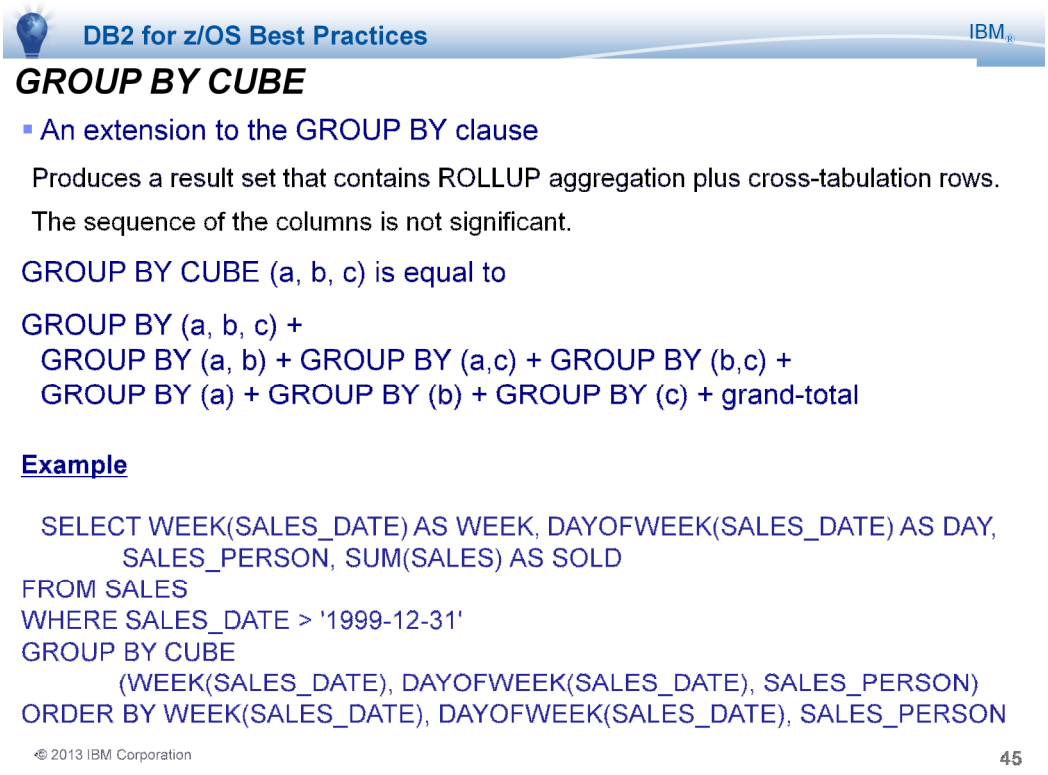
Animation is used to identify the steps (Clicks)

1.After all the sales person summaries there is a sub-total for the day, (Red)

2.Then after all sales persons and day summaries there is a sub-total for the week. (Green example adds the results from rows 4, 8, 12 & 16 to produce Row 17) 44

3.Finally there is a grand total after all week, day and sales person summaries have been calculated. (Blue shows sub-total rows 17 & 22 producing the grand total Row 23)

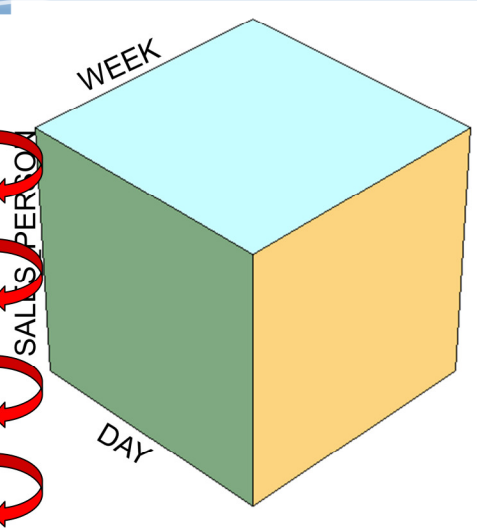
The output result set has grown from 15 rows, in the initial GROUP BY, to 23 rows because we have added the subtotal rows.



A CUBE group is another extension to the GROUP BY clause that produces a result set that contains all the rows
Like ROLLUP, a CUBE group can also be thought of as a series of grouping sets. In the case of a CUBE, all perm
The example in the slide is an extension to the previous example for the GROUP BY, notice that the *n* elements c

GROUP BY CUBE results

	WEEK	DAY	SALES_PERSON	SOLD
1	13	4	GOUNOT	11
2	13	4	LEE	10
3	13	4	LUCCHESSI	4
4	13	4	NULL	25
5	13	5	GOUNOT	20
6	13	5	LEE	21
7	13	5	LUCCHESSI	4
8	13	5	NULL	45
9	13	6	GOUNOT	4
10	13	6	LEE	27
11	13	6	LUCCHESSI	1
12	13	6	NULL	32
13	13	7	GOUNOT	14
14	13	7	LEE	25
15	13	7	LUCCHESSI	4
16	13	7	NULL	43
17	13	NULL	GOUNOT	49
18	13	NULL	LEE	83
19	13	NULL	LUCCHESSI	13
20	13	NULL	NULL	145
21	53	7	GOUNOT	1
22	53	7	LEE	6
23	53	7	LUCCHESSI	1
24	53	7	NULL	8



In this example, the first part of the result set is shown on this slide, and the 2nd part is shown on the next slide. To calculate the CUBE results, we have replaced the ROLLUP in the previous query with CUBE in the GROUP BY clause.

The results are more readily understood if you think of them as a CUBE with the dimensions WEEK, DAY & SALES_PERSON. Full results shown below, based on this result set we can think of the answers as 3 cross tabulation tables.

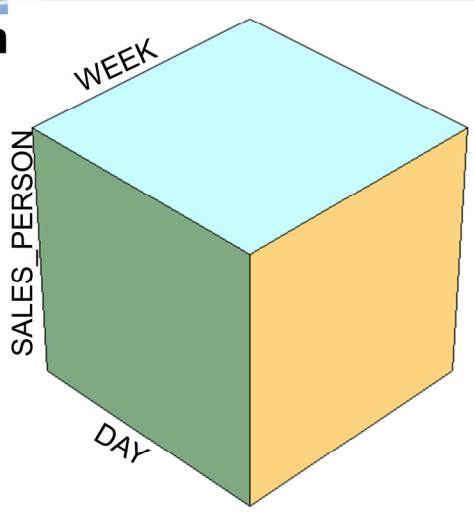
1.Units sold for each Sales Person by Day for Week 13 (RED) with sub totals by day, then sub totals for each Sales Person for the week and a total for Week 13

2.Units sold for each Sales Person by Day for Week 53 (GREEN) with sub totals by day, then sub totals for each Sales Person for the week and a total for Week 53 continued on next slide

GROUP BY CUBE results con

25	53	NULL	GOUNOT	1
26	53	NULL	LEE	6
27	53	NULL	LUCCHESSI	1
28	53	NULL	NULL	8
29	NULL	4	GOUNOT	11
30	NULL	4	LEE	10
31	NULL	4	LUCCHESSI	4
32	NULL	4	NULL	25
33	NULL	5	GOUNOT	20
34	NULL	5	LEE	21
35	NULL	5	LUCCHESSI	4
36	NULL	5	NULL	45
37	NULL	6	GOUNOT	4
38	NULL	6	LEE	27
39	NULL	6	LUCCHESSI	1
40	NULL	6	NULL	32
41	NULL	7	GOUNOT	15
42	NULL	7	LEE	31
43	NULL	7	LUCCHESSI	5
44	NULL	7	NULL	51
45	NULL	NULL	GOUNOT	50
46	NULL	NULL	LEE	89
47	NULL	NULL	LUCCHESSI	14
48	NULL	NULL	NULL	153

Total 48 records shown



47

… 2nd part of the result set from previoius slide.

1.Units sold for each Sales Person by Day for Week 53 (GREEN) with sub totals by day, then sub totals for each Sales Person for the week and a total for Week 53 continued from the previous slide

2.Units sold by each Sales Person by Day in Weeks 13 & 53(Normal with Nulls shown in the Week High-lighted with PURPLE on slide)

3.Total units sold by each Sales Person and a grand total (BLUE)

Best Practice Shops

- Keep SQL skills up to date
- Have a culture geared toward proper use of SQL
- Have SQL code review policies and procedures prior to production
- Monitor SQL performance on a regular basis
- Move the access path initial review into the developer's hands using tools – topic for another day

1. Examine Program logic – check for program filtering and joining. Move work into the query.
2. Examine FROM clause – order of tables insignificant unless > 9 table joins. List preferred join sequence for this and OUTER JOINs
3. Verify Join conditions – make sure every table is hooked up correctly to avoid cartesian joins
4. Promote Stage 2's/Residuals and Stage 1's if possible – promotions can change access paths
5. Verify data type matches – mismatched numeric and date/time will cause delays in filtering and alter the access path
6. Prune SELECT lists – remove columns with values determined to be static by WHERE clause filtering. Remove columns used in the ORDER BY or GROUP BY sequencing but not needed for the display.
7. Verify local filtering sequence – If host variables are used, add parenthesis to override the predetermined filtering sequence when necessary. This reduces the CPU required to disqualify rows
8. Analyze Access Paths – Only check the access path of the FINAL query, after query rewrite, bound with production statistics in a subsystem that resembles the production thresholds as closely as possible.
9. Tune if necessary – A topic for another day.

Recommended Reading IBM®

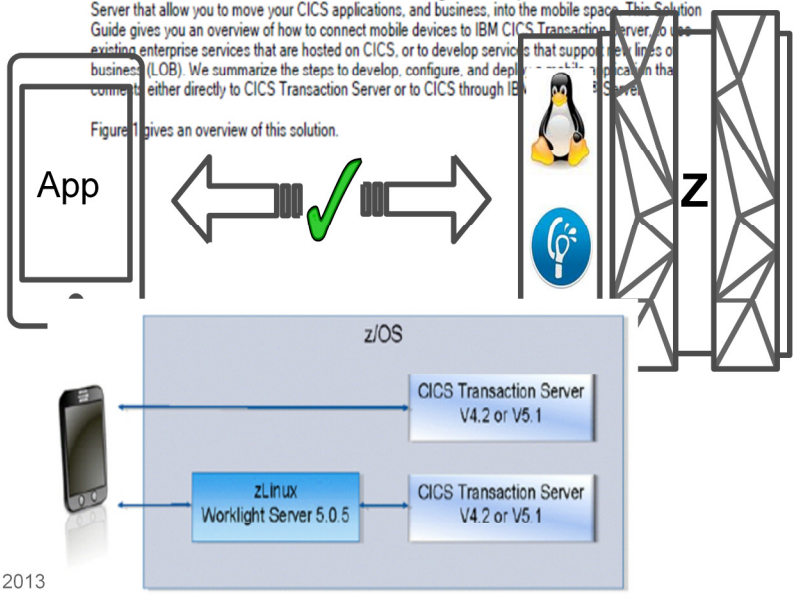
Redbooks®

Implementing IBM CICS JSON Web Services for Mobile Applications

IBM Redbooks Solution Guide

This IBM® Redbooks® Solution Guide describes the existing and new aspects of IBM CICS® Transaction Server that allow you to move your CICS applications, and business, into the mobile space. This Solution Guide gives you an overview of how to connect mobile devices to IBM CICS Transaction Server, to use existing enterprise services that are hosted on CICS, or to develop services that support new lines of business (LOB). We summarize the steps to develop, configure, and deploy a mobile application that connects either directly to CICS Transaction Server or to CICS through IBM [illegible] Server.

Figure 1 gives an overview of this solution.



1. Examine Program logic – check for program filtering and joining. Move work into the query.
2. Examine FROM clause – order of tables insignificant unless > 9 table joins. List preferred join sequence for this and OUTER JOINs
3. Verify Join conditions – make sure every table is hooked up correctly to avoid cartesian joins
4. Promote Stage 2's/Residuals and Stage 1's if possible – promotions can change access paths
5. Verify data type matches – mismatched numeric and date/time will cause delays in filtering and alter the access path
6. Prune SELECT lists – remove columns with values determined to be static by WHERE clause filtering. Remove columns used in the ORDER BY or GROUP BY sequencing but not needed for the display.
7. Verify local filtering sequence – If host variables are used, add parenthesis to override the predetermined filtering sequence when necessary. This reduces the CPU required to disqualify rows
8. Analyze Access Paths – Only check the access path of the FINAL query, after query rewrite, bound with production statistics in a subsystem that resembles the production thresholds as closely as possible.
9. Tune if necessary – A topic for another day.