DB2 9 for z/OS
Hints and Tips for Application Programmers

par Namik Hrle IBM

GUIDE Share France
Une Association Indépendante d'Utilisateurs IBM

Réunion du Guide DB2 pour z/OS France
Vendredi 27 novembre 2009
Tour Euro Plaza, Paris-La Défense

Disclaimer

© Copyright IBM Corporation 2009. All rights reserved.
U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

IBM, the IBM logo, ibm.com, DB2 and DB2 for z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

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

DB2 9 – A Rich, Features Filled Release

- SHRLEVEL(REFERENCE) for REORG of LOB tablespaces
- Online RENAME COLUMN
- Online RENAME INDEX
- Online CHECK DATA and CHECK LOB
- Online REBUILD INDEX
- Online ALTER COLUMN DEFAULT
- More online REORG by eliminating BUILD2 phase
- Faster REORG by intra-REORG parallelism
- Renaming SCHEMA, VCAT, OWNER, CREATOR
- LOB Locks reduction
- Skipping locked rows option
- Tape support for BACKUP and RESTORE SYSTEM utilities
- Recovery of individual tablespaces and indexes from volume-level backups
- Enhanced STOGROUP definition
- Conditional restart enhancements
- Histogram Statistics collection and exploitation
- WS II OmniFind based text search
- DB2 Trace enhancements
- WLM-assisted Buffer Pools management
- . . .

- Global query optimization
- Generalizing sparse index and in-memory data caching method
- Optimization Service Center
- Autonomic reoptimization
- Logging enhancements
- LOBs network flow optimization
- Faster operations for variable-length rows
- NOT LOGGED tablespaces
- Index on expressions
- Universal Tablespaces
- Partition-by-growth tablespaces
- APPEND option at insert
- Autonomic index page split
- Different index page sizes
- Support for optimistic locking
- Faster and more automatic DB2 restart
- RLF improvements for remote application servers such as SAP
- Preserving consistency when recovering individual objects to a prior point in time
- CLONE Table: fast replacement of one table with another
- Index compression
- Index key randomization
- . . .

- DECIMAL FLOAT
- BIGINT
- VARBINARY, BINARY
- TRUNCATE TABLE statement
- MERGE statement
- FETCH CONTINUE
- ORDER BY and FETCH FIRST n ROWS in sub-select and full-select
- ORDER OF extension to ORDER BY
- INTERSECT and EXCEPT Set Operations
- Instead of triggers
- Various scalar and built-in functions
- Cultural sort
- LOB File Reference support
- XML support in DB2 engine
- Enhancements to SQL Stored Procedures
- SELECT FROM UPDATE/DELETE/MERGE
- Enhanced CURRENT SCHEMA
- IP V6 support
- Unified Debugger
- Trusted Context
- Database ROLEs
- Automatic creation of database objects
- Temporary space consolidation
- 'What-If' Indexes
- . . .

TCO Reduction **Continuous Operations** **Performance** **Scalability** **SQL** **Portability**

SELECT FROM UPDATE/DELETE/MERGE

V8

SELECT FROM INSERT

Retrieves columns values created by INSERT in a single SELECT statement including:

- Identity columns
- Sequence values
- User-defined defaults
- Expressions
- Columns modified by BEFORE INSERT trigger
- ROWIDs

Avoids possible expensive access path that separate SELECT might be using

V9

SELECT FROM INSERT
UPDATE
DELETE
MERGE

One SQL call to DB2 modifies the table contents and returns the resultant changes to the application program.

E.g. we can now code destructive read from a table when a SELECT FROM DELETE statement is included. This feature is particularly useful when a table is used as a data queue.

ORDER BY and FETCH FIRST in Subselect

V8

ORDER BY and FETCH FIRST can be specified only as part of select-statement, i.e. one can write:

```
SELECT * FROM T1
  ORDER BY C1
  FETCH FIRST 1 ROW ONLY
```

but not the following:

```
INSERT INTO T2
 (SELECT * FROM T1
   ORDER BY C1
   FETCH FIRST 1 ROW ONLY)
```

V9

The restriction has been removed – the clauses can be specified in either a subselect or fullselect.

Interesting example: a loop over the statement followed by a commit deletes rows without acquiring too many locks

```
DELETE FROM  T1 X
  WHERE EXISTS
    (SELECT * FROM T1 Y
       WHERE X.KEY1=Y.KEY1
           AND X.KEY2=Y.KEY2
           AND delete_predicate
        FETCH FIRST 10000
                          ROWS ONLY
```

Skipping Locked Rows

V8

Application does not scale well due to increased lock contention.
&
Application semantics requires committed and **available** rows only.

An example of such an application is a messaging system:

- only committed and available messages can be processed
- those locked at the time will be processed later.

V9

New SQL clause: SKIP LOCKED DATA

Applies to:

- select-statements, SELECT INTO, PREPARE, searched UPDATE, searched DELETE, UNLOAD utility
- Isolation levels CS or RS
- Row or page level locking

It allows a transaction to skip over rows that are incompatibly locked by other transactions, without being blocked.

An interesting usage scenario is serializing access to any kind of object:

- Create a table and insert a row for each object to be controlled
- Code: SELECT … FOR UPDATE OF … SKIP LOCKED DATA
- The object unavailability is identified by return code +100 (without any wait)

New Techniques to Retrieve LOBs: FETCH CONTINUE

V8

Existing techniques to retrieve entire LOBs with a large maximum length are not optimal

- Fetch into pre-allocated buffer
  - using max size buffer results in best performance, but inefficient storage utilization
  - Using smaller buffer results in truncation which is not accompanied by the actual LOB size
- Using LOB locators
  - storage efficient but requires multiple trips to DB2

V9

FETCH WITH CONTINUE when retrieving base row followed by
FETCH CURRENT CONTINUE if LOB is truncated

New techniques to retrieve LOBs:

1. Fetch into moderately sized buffer, expected to fit most values. If it does not:
   - Allocate larger buffer using the actual LOB length returned on the FETCH
   - FETCH CURRENT CONTINUE to retrieve the rest
2. Fetch through a streaming, relatively small buffer (e.g. 32KB)
   - move and reassemble data pieces into a larger buffer or pipe it to an output file

These techniques apply to XML as well

LOBs Network Flow Optimization

V8

- LOBs retrieval optimized for large amounts of data in which case LOB locators usage is the most efficient technique.
- However, for smaller LOBs, returning the LOB value directly is more efficient.
- Therefore, the applications use different techniques depending on the actual LOB size. E.g. in Java via JCC property settings:
  - materialized LOB fullyMaterializeLobData=true (default)
  - usage of LOB locator fullyMaterializeLobData=false
- Locators remain active for the scope of the transaction (unless explicitly freed) and valuable server resources are longer held.

V9

- Progressive streaming for LOB/XML
- Behavior depends on LOB/XML size:
  - size < 32KB, data is in-lined (like varchar)
  - 32k < size < 1MB, data is chained to the query result
  - size > 1MB, LOB locator is returned
- For Java, activated by:
  - Setting connection property progressiveStreaming=ON
  - Using either:
    - LOB streaming interface, or
    - LOB object interface
- For CLI, activated by using streaming interface SQLGetData
- Requires DB2 Connect 9.1 FP 1, better yet 9.5

New Data Types: BINARY and VARBINARY

- BINARY fixed-length binary string
  - 1 to 255 bytes
- VARBINARY variable-length binary string
  - 1 to 32704 bytes; maximum length determined by the maximum record size associated with the table
- Unlike FOR BIT DATA the new BINARY and VARBINARY use x'00' as a padding character
- Comparison rule:
  - If two strings are equal up to the length of the shorter string , the shorter string is considered less than the longer string.

New Data Type: DECFLOAT

V8

Binary fractions cannot exactly represent most decimal fractions (e.g. 0.1 requires an infinitely long binary fraction)

So, is 1.2 x 1.2 = 1.44?
1.2 in a 32-bit binary float

- 1.2 in a 32-bit binary float is actually
  1.200000047683715820312 5
- … and this squared is
  1.44000005722045898437 5

V9

New DECFLOAT data type is well suited to typical customer financial calculations, similar to “calculator” mathematics.

Eliminates rounding errors by using base 10 math and provides floating point convenience with fixed point precision.

Has up to 34 digits of precision

DECFLOAT(16)
10+384 to 10-383 positive & negative

DECFLOAT(32)
10+6144 to 10-6143 positive & negative

New Data Type: BIGINT

V8

BIGINT (big integer) is not supported as a native data type

Simulated by DECIMAL(19,0)

V9

- BIGINT is newly supported data type and function
- Like SMALLINT and INTEGER, this is an exact numeric type
- 63-bit precision
  - From - 9223372036854775808
  - To + 9223372036854775807
- Compatible with all numeric data types
- The BIGINT function returns a big integer representation of a number or a string representation of a number

APPEND

V8

All of the following applies:

- Critical, high insert rate workload needs better performance and all the conventional tuning steps have already been applied.
- Clustering is either not beneficial or more frequent reorganizations are acceptable
- *MC00* insert algorithm is still not fast enough or the prerequisites cannot be satisfied:
  - MEMBER CLUSTER
  - FREEPAGE=PCTFREE=0

V9

CREATE TABLE … APPEND YES | NO

ALTER TABLE … APPEND YES | NO

- The APPEND YES results in a fast insert at the end of the table or appropriate partition at the expense of data organization and rapid table space growth.
- After populating with the APPEND option in effect, clustering can be achieved by running the REORG utility providing a clustering index has been explicitly defined.
- Make sure PK81471 is applied
- Note that MC00 is still valid, but make sure that PK81470 is applied

Virtual Indexes a.k.a. ‘What If’ Indexes

V8

How to determine that a new index would benefit a given **dynamic SQL** query?

How to determine that dropping an index will not negatively affect a given query?

- In many cases predicting based on modeling is not reliable due to query complexity
- Indiscriminate adding of indexes creates permanent overhead for most operations (SQL and utilities)
- Creating a new index is obtrusive for concurrent operations
- Using a test system for experimenting lacks potentially crucial environmental factors that affect access path selection

V9

- Virtual i.e. hypothetical indexes can be specified and made visible to statement EXPLAIN STATEMENT FOR
- Table DSN_VIRTUAL_INDEXES is used to specify virtual indexes
  - Table columns include selected columns from SYSINDEXES and SYSKEYS
  - Users need to create the table manually, unless tooling such as Index Advisor does it automatically. Appropriate script is provided.
  - To create/drop an index, the table needs to be populated with a row that provide an appropriate description of index
- At EXPLAIN time, during query optimization, the virtual indexes compete with regular indexes on the tables in a cost-based fashion and the dropped indexes are not considered

DSN_VIRTUAL_INDEXES Table

TBCREATOR	Authorization ID of owner (or schema in V9) of table on which the index is being created/dropped
TBNAME	Name of the table on which the index is being created or dropped
IXCREATOR	Authorization ID (or schema in V9) of the owner of the index
IXNAME	Name of the index
ENABLE	Whether this index specification will be processed ('Y') or not ('N').
MODE	Whether the index is being created ('C') or dropped ('D')
UNIQUERULE	Whether the index is unique: D for No (duplicates are allowed); U for Yes
COLCOUNT	The number of columns in the key
CLUSTERING	Whether the index is clustered ('Y' or 'N')
NLEAF	Number of active leaf pages in the index. If unknown, the value must be -1.
NLEVELS	Number of levels in the index tree. If unknown, the value must be -1.
INDEXTYPE	The index type: '2' - NPSI; 'D' - DPSI
PGSIZE	Size, in bytes, of the leaf pages in the index: 4K, 8K, 16K, 32K
FIRSTKEYCARDF	Number of distinct values of the first key column. If unknown, the value must be -1.
FULLKEYCARDF	Number of distinct values of the key. If unknown, the value must be -1.
CLUSTERRATIOF	Clustering ratio. . If unknown, the value must be -1.
PADDED	Indicates whether keys within the index are padded for varying-length column data ('Y' or 'N')
COLNO1	Column # of the first column in the index key
ORDERING1	Ordering ('A' or 'D') of the first column in the index key
...	...
COLNO64	Column # of the last column in the index key. Needs to be populated only when # index keys = 64
ORDERING64	Ordering ('A' or 'D') of the last column in the index key.

TRUNCATE TABLE

V8

Alternative way of deleting the entire table is needed for any of these reasons:

- DELETE without WHERE clause is not fast enough as the table includes delete triggers
- Using LOAD REPLACE with empty input data set (even when called via DSNUTILS) is not DBMS agnostic
- Storage occupied by deleted rows should be released faster

V9

New DML statement:

TRUNCATE table
DROP | REUSE STORAGE
IGNORE | RESTRICT DELETE TRIGGERS
IMMEDIATE

Under the cover it's DELETE without WHERE clause, but without delete triggers processing overhead.
Therefore it is fast for tables in segmented and universal tablespaces for which there are no CDC, MLS and VALIDPROC enabled attributes.

MERGE Statement

V8

For a set of input rows update the target table when the key exists and insert the rows for which keys do not exist.
E.g.

- For activities whose description has been changed, update the description in table **archive**.
- For new activities, insert into **archive**.

Prior to V9 this has been coded as a loop over conditional INSERT and UPDATE statements

V9

```
MERGE INTO archive AR
  USING VALUES (:hv_activity, :hv_description) FOR :hv_nrows ROWS
    AS AC (ACTIVITY, DESCRIPTION)
  ON (AR.ACTIVITY = AC.ACTIVITY)
  WHEN MATCHED THEN UPDATE SET DESCRIPTION = AC.DESCRIPTION
  WHEN NOT MATCHED THEN INSERT (ACTIVITY, DESCRIPTION)
                                VALUES (AC.ACTIVITY, AC.DESCRIPTION)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

MERGE Example

```
MERGE  INTO account AS T
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS AS S (id, amt)
ON T.id = S.id
WHEN MATCHED          THEN UPDATE SET balance = T.balance + S.amt
WHEN NOT MATCHED   THEN INSERT (id, balance) VALUES (S.id, S.amt)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

MERGE

VALUES AS S

id	amt
1	30
5	10
10	40
5	20
1	50

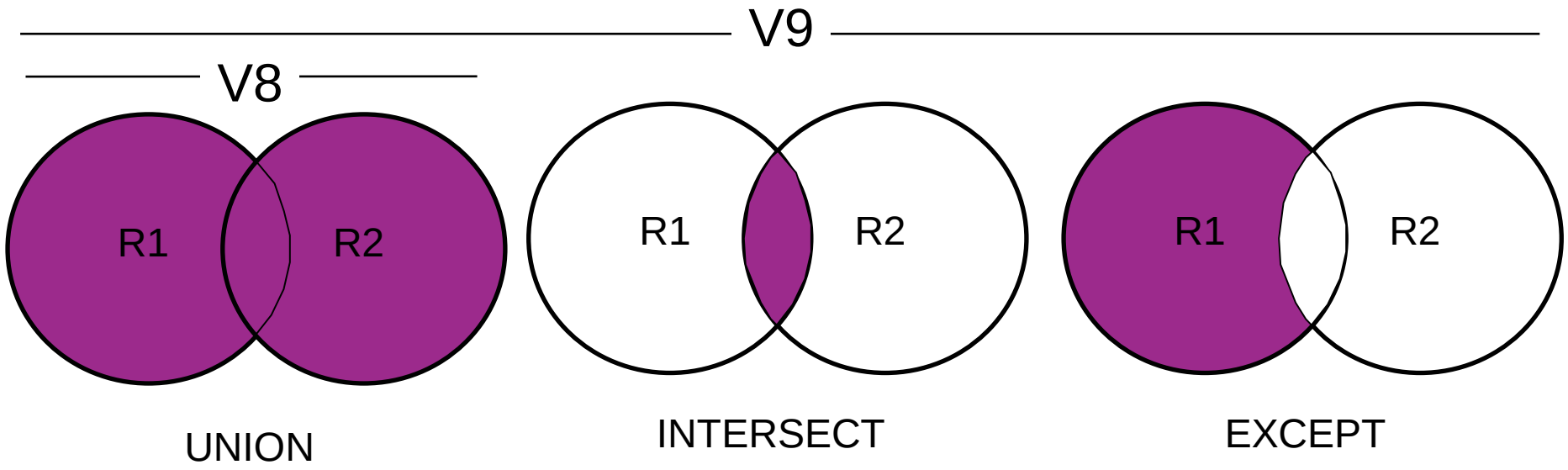
account AS T

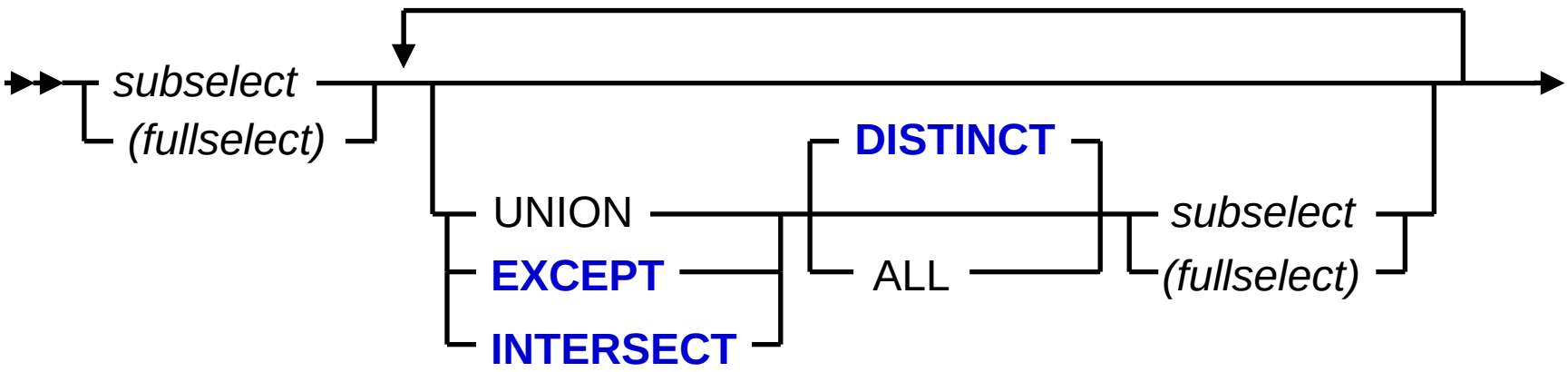
id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000

account AS T

id	balance
1	1080
5	30
10	540
200	600
300	300
315	100
500	4000

INTERSECT/EXCEPT





INSTEAD OF Triggers

V8

- No unified mechanism for controlling read and write access by an application (e.g. encryption)
  - Views are used for read access control
  - Triggers on base table are used for write access control
- No INSERT/UPDATE/DELETE for read-only views (e.g. joins)

V9

- A new type of trigger (in addition to BEFORE and AFTER triggers)
- Can only be defined on views
- DB2 only executes the *triggered-action* instead of the action against the subject view
  - application still believes all operations are performed against the view
- Provides means to update views that are considered read-only by DB2

INSTEAD OF TRIGGERS

```
CREATE TABLE WEATHER (CITY VARCHAR(25), TEMPF DECIMAL(5,2))
CREATE VIEW CELCIUS_WEATHER (CITY, TEMPC) AS
      SELECT CITY, (TEMPF-32)*5.00/9.00 FROM WEATHER

CREATE TRIGGER CW_INSERT INSTEAD OF INSERT ON
        CELCIUS_WEATHER
REFERENCING NEW AS NEWCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
    INSERT INTO WEATHER VALUES (NEWCW.CITY, 9.00/5.00*NEWCW.TEMPC+32)

CREATE TRIGGER CW_UPDATE INSTEAD OF UPDATE ON
        CELCIUS_WEATHER
REFERENCING NEW AS NEWCW OLD AS OLDCW DEFAULTS NULL
FOR EACH ROW MODE DB2SQL
    UPDATE WEATHER AS W
      SET W.CITY = NEWCW.CITY,
            W.TEMPF = 9.00/5.00*NEWCW.TEMPC+32
      WHERE W.CITY = OLDCW.CITY
```

Support for Optimistic Locking

V8

- With optimistic locking the retrieved rows are not protected by locks after they are retrieved.
- That means they can be changed by concurrent transactions after the retrieval and before they are to be updated by the transaction that retrieved them in the first place.
- In order to ensure consistency, the retrieved rows must be checked if they changed since they had been retrieved.
- Prior to V9 all of the columns marked for update and their values in last read operation are added explicitly in the WHERE clause of the UPDATE, so that the UPDATE fails if the underlying column values have been changed.

Support for Optimistic Locking

V9

- V9 adds a new expression: ROW CHANGE TOKEN which returns a token that represents a relative point in the modification sequence of a row
- Instead of knowing all the old values that are to be updated, the application can compare the current ROW CHANGE TOKEN value of a row with the value that was stored when the row was last time fetched.
- ROW CHANGE TOKEN is generated in two alternative ways:
  - From an optional, explicit (but optionally hidden) column
  - From the page RBA or LRSN

```
SELECT C1, ROW CHANGE TOKEN FOR TAB, RID(TAB)
  INTO :h1, :h_rct. :h_rid
  FROM TAB WHERE TAB.C1 = 10
… some other statements in the application …
UPDATE TAB
  SET TAB.C2 = 10
  WHERE RID(TAB) = :h_rid AND ROW CHANGE TOKEN FOR TAB = :h_rct
```

LOB Performance and Concurrency Enhancements

V8

Very large number of locks even for UR scanners resulting in a high IRLM storage usage or lock escalations

The reason is LOB locks which are acquired for any data access operation in order to:

- control LOB space usage
- serialize readers and updaters of LOB columns

V9

- For UPDATE, INSERT and DELETE LOB lock avoidance will be attempted. If it fails, the resulting X-LOB lock will have *manual* duration only (unlock after the operation completion). Changed LOB data pages and its index pages are flushed out to the GBPs prior to the unlock in order to ensure consistency for UR readers on other data sharing members.
- For non-UR SELECTs LOB lock will be no longer acquired
- For UR SELECTs the resulting S-LOB locks will have *autorel* mode (lock is acquired and released immediately)
- LOB locks as means to control space reuse are no longer used. DB2 relies on other technique to achieve that.

Index on Expression

V8

How to improve performance of queries that include expressions such as in the following examples:

```
SELECT id
  FROM employee
  WHERE
    UPPER (lastname, 'EN_US') = 'JOE'
    AND
    UPPER (firstname, 'EN_US') = 'JOHN'
```

```
SELECT id
  FROM employee
  WHERE
    bonus + salary > 100000
```

V9

Indexes on expressions:

```
CREATE INDEX upper_empname
   ON employee
   (UPPER (lastname, 'EN_US'),
    UPPER (firstname, 'EN_US'))
```

```
CREATE INDEX total_compensation
   ON employee
   (salary + bonus)
```

- Extra cost in Load, Insert, Update on key value, Rebuild Index, Check Index, and Reorg Tablespace, but not Reorg Index
- Not eligible for zIIP offload
- The rules for creating indexes on expression are more restrictive than for traditional indexes

Miscellaneous

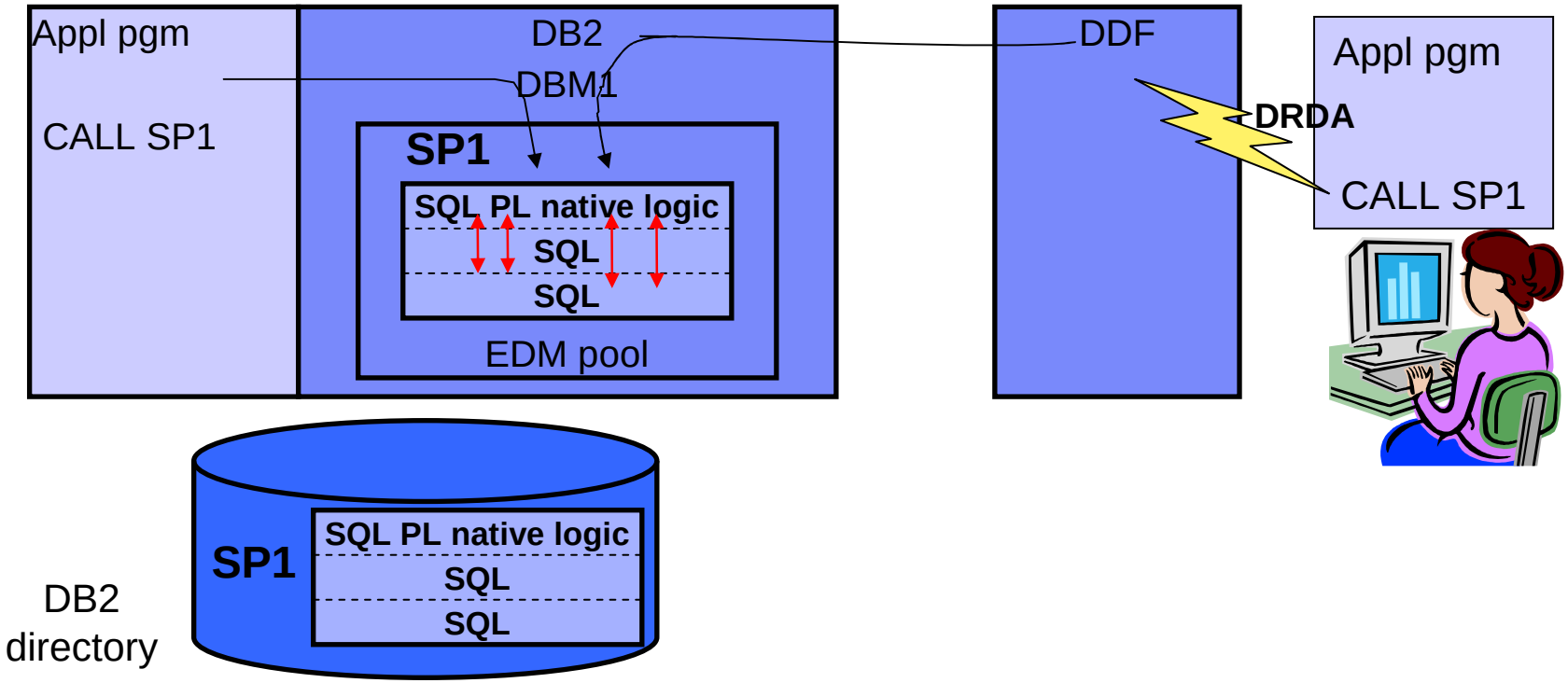
- LOB File Reference support
  - A file reference variable allows direct transfer of LOB data between DB2 and the file named in the variable
- The RELCURHL = NO zparm option is removed
  - Very unlikely incompatibility possibility for applications dependent on retaining page or row locks across commits for a WITH HOLD cursor
- Faster operations for variable-length rows
  - Remember tuning recommendations for rows with variable-length columns?
  - New, **R**eordered **R**ow **F**ormat

Prefix	Fixed Length Cols	Varchar Pointers	Varying Length Cols

- Index key randomization
- Enhanced CURRENT SCHEMA
  - Removing the V8 restriction that disallows CREATE statements when the value of CURRENT SCHEMA was different from the value in CURRENT SQLID
- Online ALTER TABLE RENAME COLUMN source-column-name TO target-column-name
  - Not allowed if column referenced in a view or has a trigger defined on it
- Online RENAME INDEX
- Online ALTER TABLE ALTER COLUMN SET DEFAULT
- Online ALTER TABLE ALTER COLUMN DROP DEFAULT
  - PK56392

Native Support for SQL Procedures

- Eliminates implicitly generated C code and compilation
- Fully integrated into the DB2 engine
  - An SQL procedure created without FENCED or EXTERNAL is a native SQL procedure



More Stored Procedures Enhancements

- Changing name resolution within a procedure body
- Using delimited identifiers, including lowercase characters, for SQL condition names, SQL labels, SQL variables, and SQL parameters
- Full support for nested compound statements including:
  - Use a compound statement within the body of a condition handler
  - Use nested compound statements to define different scopes for SQL variables, cursors, condition names, and condition handlers
- Versioning and managing source code
  - VERSION keyword on CREATE PROCEDURE
  - CURRENT ROUTINE VERSION special register
  - ALTER ADD VERSION, REPLACE VERSION, ACTIVATE VERSION
  - BIND PACKAGE with new DEPLOY keyword
    - Allows to deploy from test to production without doing a CREATE PROC statement
- Deploying of native SQL procedures to multiple servers
- Debugging of native SQL procedures

Seamless Integration of XML and Relational Data

V8

Two ways of simulating support for XML data

- XML view of relational data
  - Shredding and composition
- XML documents as monolithic entities
  - Atomic storage and retrieval
  - Basic search capabilities

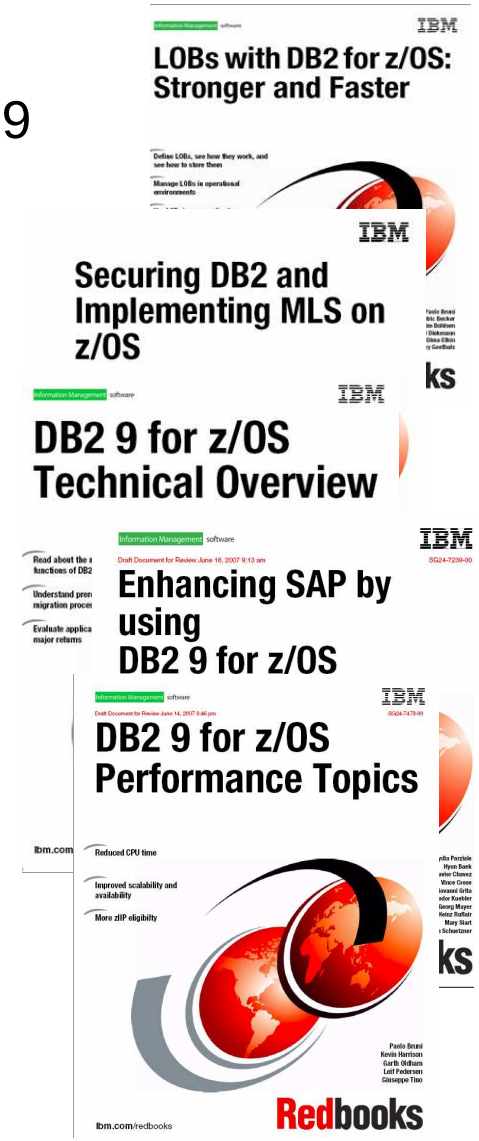
No full integration into the database system

V9

- XML document storage
  - XML as a native data type
  - Supported by most SQL operations
  - Decomposition stored procedure
- XML document retrieval
  - SQL for entire documents
  - XPath expressions through SQL/XML for portions of documents
  - Performance benefits through indexing support
- Application development support
  - Java, C/C++, .NET, PHP, COBOL, PL/1 etc.
- Database administration support
  - XML Schema Repository
  - DB2 Utilities

DB2 9 for z/OS RedBooks & RedPapers

- Powering SOA with IBM Data Servers SG24-7259
- LOBs with DB2 for z/OS: SG24-7270
- Securing DB2 & MLS z/OS SG24-6480-01
- DB2 9 Technical Overview SG24-7330
- Enhancing SAP - DB2 9 SG24-7239
- Best practices SAP BI - DB2 9 SG24-6489-01
- DB2 9 Performance Topics SG24-7473
- DB2 9 Optimization Service Center SG24-7421
- Index Compression with DB2 9 for z/OS paper
- DB2 9 Stored Procedures SG24-7604



DB2 9 for z/OS
Hints and Tips for Application Programmers

par Namik Hrle IBM
hrle@de.ibm.com