DB2 V8/V9 SQL Challenges and Solutions

Susan Lawson

Yevich, Lawson & Assoc. Inc.
2743 S. Veterans Pkwy PMB 226
Springfield, IL 62704

www.ylassoc.com
www.db2expert.com

Disclaimer PLEASE READ THE FOLLOWING NOTICE

- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an "As is" basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client's ability to evaluate and integrate them into the client's operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.

Abstract

In this seminar we will look at some of the performance and availability features in V8 and how they have been successfully used in our applications and databases to meet some of our current performance and availability challenges. We will discuss some of the things we still desire in terms of application and database performance and take a look at V9 to see what features are available to further improve our performance and availability of our applications and databases. We will look at how we may take advantage of those features.

This presentation was developed by Susan Lawson and Dan Luksetich of YLA. They can be reached at Susan_Lawson@ylassoc.com and Dan_Luksetich@ylassoc.com repectively.

Objectives

- Look at some very useful V8 features
  - Database
  - Application
  - SQL
- Discuss some current challenges with V8
  - Features that work well
  - Features that did not work so well
- Discuss some needs that still exist
  - More optimization features
  - Database availability
- Look at V9 features
  - Database
  - Application
  - SQL
- Discuss usage of new database, application and SQL features of V9
  - Look at how some of these new features will solve old problems
  - Look at new performance opportunities

Outline

- V8 SQL/Application Performance Features
  - Multi-Row Fetch
  - Recursion
  - Common Table Expressions
  - INSERT within a SELECT
  - Scalar Fullselect
  - Distribution Statistics
  - XML Scalar Functions
  - Multiple Distinct
  - Matching Predicates
  - Dynamic SQL Improvements
- V8 Database Performance Features
  - Volatile Tables
  - Data Partitioned Secondary Indexes
  - Materialized Query Tables

Outline (cont.)

- V9 SQL/Application Features
  - UPDATE/DELETE with a SELECT
  - Optimistic Locking
  - Index on Expression
  - Histogram Statistics
  - Order By and Fetch First in Subselect
  - Instead of Triggers
  - INTERSECT and EXCEPT
  - Skip Locked Data
  - Caseless Expressions
  - Native SQL Procedures
  - XML Support
  - New Functions
  - Distinct Sort Avoidance

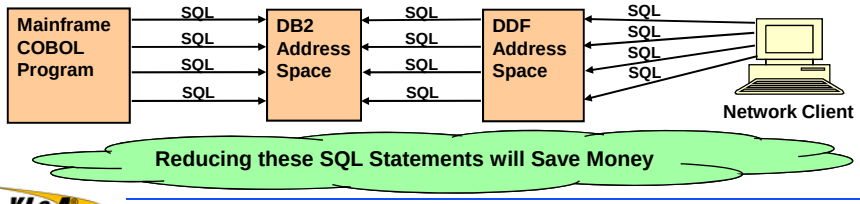
Outline (cont..)

- V9 Database Performance Features
  - Better Index Page Splitting
  - Index Compression and Large Pages
  - DPSI Enhancements
  - Clone Tables
  - Append on Insert
  - Truncate Table
  - No Log Tables

SQL and Application
Challenges and V8/V9 Solutions

Major SQL Performance Challenge

- **Generic application design has led to an increase in SQL statements issued**
  - **Every SQL call represents overhead**
    - **Cross memory communication on the mainframe**
    - **Cross network traffic plus cross memory communication from remote clients**
  - **Modern generic techniques increases the amount of SQL**
    - **Use of system generated keys**
    - **Programmatic joins**
    - **Flexible object-relational design**
- **Every call to DB2 represents CPU and elapsed time consumed**
  - **Can we reduce this time with DB2 for z/OS V8 SQL?**



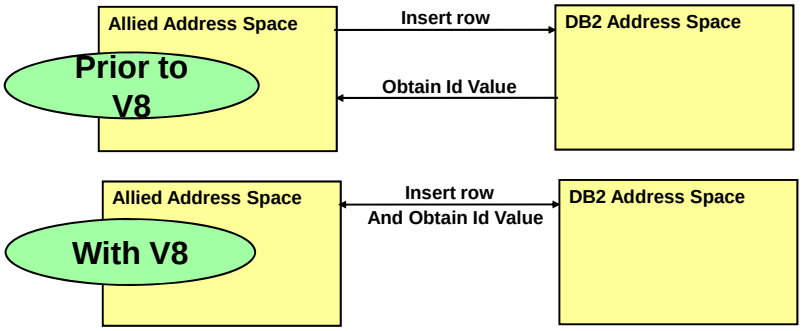
V8 - SELECT from INSERT

Find value of Identity Column during insert (assume table created with identity column on ACCT_ID generated always)

```
SELECT ACCT_ID
FROM FINAL TABLE
 (INSERT INTO UID1.ACCOUNT (NAME, TYPE, BALANCE)
        VALUES ('Master Card', 'Credit', 50000) )
```

- This feature is useful for reducing the number of times an application must travel to the database when establishing new data
  - Quantity of SQL is a major contributor to elapsed and CPU time
- Several generated data can be selected
  - Identity column values
  - ROWID
  - Sequence values
  - Defaulted values
  - Data values resulting from triggers
- Identity values generate/selected can then be used for inserting into child tables

V8 - SELECT from INSERT – for Performance



- SELECT from INSERT works very well with the V8 sequence objects

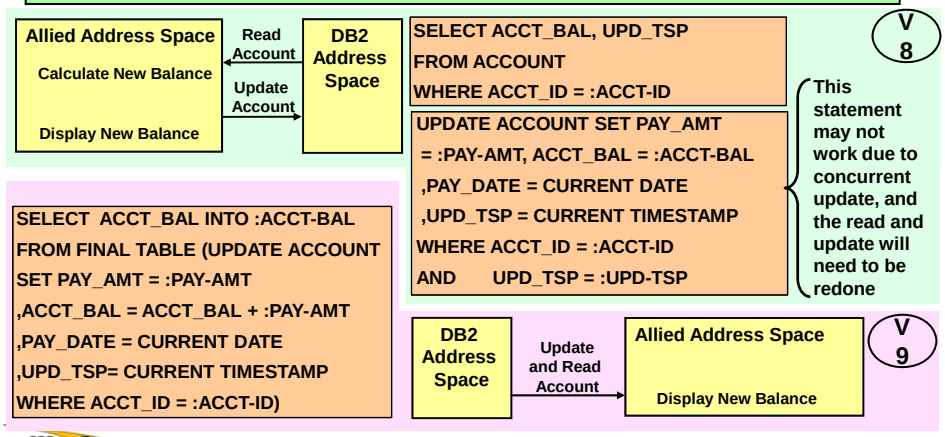
Find value of sequence object during insert

```
SELECT ACCT_ID
FROM FINAL TABLE
 (INSERT INTO UID1.ACCOUNT (ACCT_ID,NAME, TYPE, BALANCE)
  VALUES (NEXT VALUE FOR ACCT_SEQ, 'Master Card', 'Credit', 50000) )
```

V9 - SELECT from an UPDATE/DELETE

- Will allow a searched UPDATE or DELETE to be placed in the FROM clause
  - In a Cursor or in the SELECT INTO statement
  - Returning old or new data
- Reduces SQL statements issued and improves concurrency!

What if I update ACCOUNT with today's payment, and return the balance?



V9 - SELECT from an UPDATE/DELETE

- New INCLUDE clause in version 9 allows for more elimination of SQL statements replaced by single complex statement
  - Allows additional columns to be provided in the SELECT list
  - These additional columns are not included in the operation

What if I update ACCOUNT with today's payment, and return both the previous and the new balance?

```
SELECT  ACCT_BAL, OLD_BAL                          <- New column returned here
 INTO :NEW-ACCT-BAL, :ACCT-OLD-BAL
FROM FINAL TABLE
(UPDATE ACCOUNT INCLUDE(OLD_BAL DEC(9,2))          <- New column defined here
SET PAY_AMT = :PAY-AMT
    ,ACCT_BAL = ACCT_BAL + :PAY-AMT
    ,PAY_DATE = CURRENT DATE
   ,UPD_TSP= CURRENT TIMESTAMP
   ,OLD_BAL = ACCT_BAL                             <- New column value set here
WHERE ACCT_ID = :ACCT-ID)
```

V9 - SELECT from an UPDATE/DELETE

- You can SELECT from a DELETE the same as from an INSERT or UPDATE
  - The OLD TABLE clause gets the deleted row(s)
    - This clause also works for UPDATE

What if the ACCOUNT is being closed and deleted, but I wanted to return the account information?

```
SELECT  ACCT_BAL, PAY_DATE
 INTO :ACCT-BAL, :PAY-DATE
FROM OLD TABLE
(DELETE FROM ACCOUNT
WHERE ACCT_ID = :ACCT-ID)
```

V8 - Scalar Fullselect

```
SELECT COLA,
    (SELECT COLB FROM TABLE2 WHERE COLB1 = T1.COLA),
    (SELECT COLC FROM TABLE3 WHERE COLC1 = T1.COLA)
FROM TABLE1 T1
```

- The SQL language is becoming more orthogonal
  - Any expression can be placed anywhere in a SQL statement
  - A SQL statement is an expression
    - A SELECT statement
    - A WHERE clause
    - A CASE statement
- Scalar fullselects within a SQL statement can be in many ways a performance advantage!
  - Reduction of SQL statements issued
  - Utilizing correlation for transaction performance
  - Creating complex SQL statements

V8 - Scalar Fullselect – Reduction of SQL

```
SELECT COLA
FROM TABLE1 T1
WHERE COLB BETWEEN
    (SELECT MIN(COL1) FROM TABLE1)
    AND
    (SELECT MAX(COL1) FROM TABLE1)
```

Instead of three queries to get an answer, only one query is issued. Scalar fullselect provides this solution

- We can now use SQL to reduce the number of times we go to the database
  - Quantity of SQL is a major contributor to elapsed and CPU time
  - Are you using multiple queries to get data from the database?
    - You should investigate getting those answers into a single query if you want to save CPU
- Getting answers into a single SQL statement makes the SQL more portable as well as improving performance

V8 - Scalar Fullselect – Correlation

30 secs.

```
SELECT DISTINCT A.CUST_ID,
            B.BUNDLE_FLG,
FROM   CUST_PRDT_PRC_SUMM A
LEFT   OUTER JOIN
(SELECT DISTINCT CUST_ID,
             1 as BUNDLE_FLG
 FROM    CUST_PRC_SUMM_CMPT) B
 ON      A.CUST_ID = B.CUST_ID
```

This query performs an existence check on the CUST_PRC_SUMM_CMPT table, but the table expression is materialized due to DISTINCT and the entire table is read

<1 sec.

```
SELECT DISTINCT A.CUST_ID,
  (SELECT MAX(CUST_ID)
   FROM   CUST_PRC_SUMM_CMPT B
   WHERE  A.CUST_ID = B.CUST_ID) AS BUNDLE_FLG
FROM CUST_PRDT_PRC_SUMM A;
```

This query returns the same result, but correlation is used to probe the inner table and avoid materialization and using an index on CUST_ID

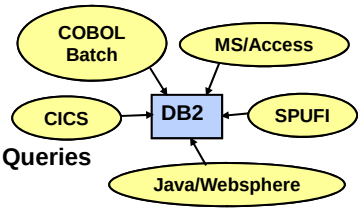
- Use correlation for transactions when outer table processes relatively little data
  - Encourages nested loop join
  - Encourages index access

V8 - Scalar Fullselect – Complex SQL

```
SELECT ACCT_NUM, ACCT_RATING_CDE,
    (SELECT MAX(DOLLAR_AMT)
     FROM   ACCT_HIST B
     WHERE  B.ACCT_ID = A.ACCT_ID)
FROM   ACCT_ID A
WHERE  ACCT_ID = 543670001
```

This query returns both line item and aggregate data in one statement!

- Complex SQL
  - Performance Improvement
    - Data is Filtered or Aggregated
    - Transactions Process Little Data
    - One Large Query Instead of Many Small Queries
    - Little or No Logic in SELECT List
      - Avoid UDFs, CASE Expressions, Data Conversions
  - Flexibility
    - SQL is Highly Portable
    - Relatively Easy to Code – Fast Time to Delivery
    - Write a SQL statement in one place, and run it anywhere



V8 - Common Table Expressions

```
WITH DEPTOTAL (DEPT_NO, MAXSALARY) AS
    (   SELECT DEPT_NO, SUM(SALARY+COMM)
        FROM EMP
        GROUP BY DEPT_NO  )
SELECT DEPT_NO
FROM DEPTOTAL
WHERE MAXSALARY =
          (   SELECT MAX(MAXSALARY)
              FROM DEPTOTAL   )
```

Common table expressions are materialized on first reference

DEPTOTAL referenced here

The WITH clause defines a table called DEPTOTAL. This table can be used within the SQL statement, including other common table expressions

- A table defined in a SQL statement that exists for the duration of that SQL statement
  - Can provide performance improvements by computing a value once, not multiple times
  - Can be used in SELECT, CREATE VIEW and INSERT
  - Can contain references to host variables
  - Simplifies complex SQL to reduce possibility of programmer error
- Common table expressions are extremely powerful and enable
  - Complex SQL
  - Reduction of SQL issued

V8 - Common Table Expression Walkthrough

```
WITH DEPTOTAL (DEPT_NO, MAXSALARY) AS

SELECT DEPT_NO, SUM (SALARY+COMM)
       FROM EMP
       GROUP BY DEPT_NO

SELECT DEPT_NO FROM DEPTOTAL
WHERE MAXSALARY =

       (SELECT MAX(MAXSALARY
              FROM DEPTOTAL)
```

DEPT_NO	MAXSALARY
1	21,000
2	32,000

DEPT_NO
2

Final Result

MAXSALARY
32,000

EMP

DEPT_NO	SALARY	COMM
1	10,000	1,000
1	20,000	1,000
2	10,000	1,000
2	30,000	2,000

V8 - CTE – Reduction of SQL - Max Performance

- Prior to V8 searching on multiple values was difficult

V7

```
SELECT PERSON_ID
FROM   PERSON_TBL
WHERE  (LASTNAME = 'SMITH'
        OR LASTNAME = 'JONES')
AND    FIRST_INIT = 'A'
```

Using an index on LASTNAME, FIRST_INIT, this statement can only index match on LASTNAME

V7

```
SELECT PERSON_ID
FROM   PERSON_TBL
WHERE  LASTNAME = 'SMITH'
AND    FIRST_INIT = 'A'
UNION ALL
SELECT PERSON_ID
FROM   PERSON_TBL
WHERE  LASTNAME = 'JONES'
AND    FIRST_INIT = 'A'
```

- V8 Gives us more choices

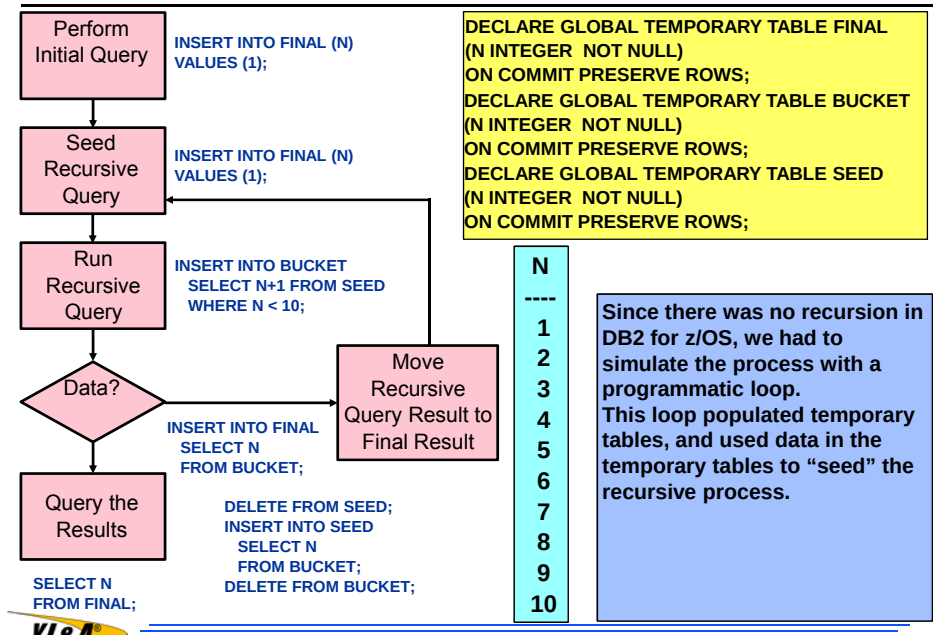
V8

```
WITH PRSN_SEARCH(LASTNAME) AS
(SELECT 'SMITH' FROM SYSIBM.SYSDUMMY1
UNION ALL
SELECT 'JONES' FROM SYSIBM.SYSDUMMY1)
FROM   PERSON_TBL A, PRSN_SEARCH B
WHERE  A.LASTNAME = B.LASTNAME
AND    FIRST_INIT = 'A'
```

This statement matches on both indexed columns!

This query may use both columns of the index, but we still probe twice. This could also be in multiple statements!

Simulating Recursion in DB2 z/OS Prior to V8



V8 - Recursion Example

```
WITH TEMP(N) AS
  (SELECT 1
   FROM SYSIBM.SYSDUMMY1
  UNION ALL
  SELECT N+1
  FROM TEMP
  WHERE N < 10)
SELECT N
FROM TEMP
```

Initial select sets the starting values

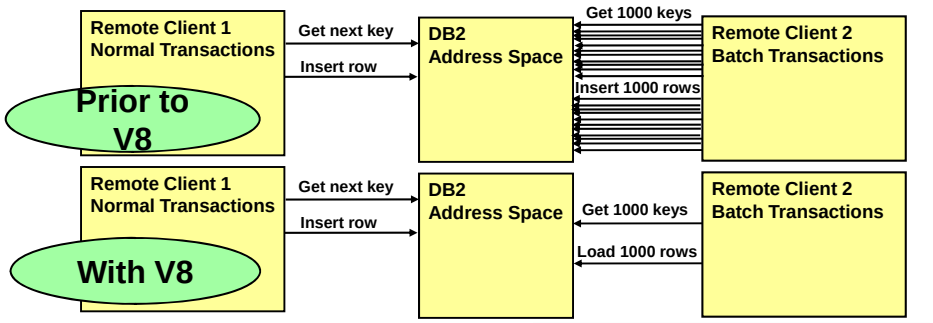
Iterative Select increments the value

Result

N
1
2
3
4
5
6
7
8
9
10

- Uses same syntax as common table expressions
- Ability to iterate in the SQL statement
- Be sure to add controls to stop
  - Else...SQL +347 The recursive common table expression "TEMP" may contain an infinite loop. SQLSTATE=01605

V8 - Recursion – Reducing SQL



- Application needed a system generated key
  - Sequence object in V8
  - Identity column in V7
- Both batch and online can insert rows
  - Batch is high volume
  - message traffic was killing performance
- V8 to the rescue
  - Get 1000 keys in one message

```
WITH GET_THOUSAND (N) AS
  (SELECT 1
   FROM SYSIBM.SYSDUMMY1
   UNION ALL
   SELECT N+1
   FROM   GET_THOUSAND
   WHERE  N < 1000)
SELECT NEXT VALUE FOR SEQ1
FROM   GET_THOUSAND;
```


V8 - Recursion – Generating Data

```
WITH LASTPOS (KEYVAL) AS
   (VALUES (0)
    UNION ALL
    SELECT KEYVAL + 1
    FROM   LASTPOS
    WHERE  KEYVAL < 9)
,STALETBL (STALE_IND) AS
   (VALUES 'S', 'F')
SELECT STALE_IND, KEYVAL
 ,CASE STALE_IND WHEN 'S' THEN
   CASE KEYVAL WHEN 0 THEN 1
    WHEN 1 THEN 2 WHEN 2 THEN 3
    WHEN 3 THEN 4 WHEN 4 THEN 4
    WHEN 5 THEN 6 WHEN 6 THEN 7
    WHEN 7 THEN 8 WHEN 8 THEN 9
    WHEN 9 THEN 10 END
                WHEN 'F' THEN
   CASE KEYVAL WHEN 0 THEN 11
    WHEN 1 THEN 12 WHEN 2 THEN 13
    WHEN 3 THEN 14 WHEN 4 THEN 15
    WHEN 5 THEN 16 WHEN 6 THEN 17
    WHEN 7 THEN 18 WHEN 8 THEN 19
    WHEN 9 THEN 20 END
       END AS PART_NUM
FROM  LASTPOS INNER JOIN
      STALETBL ON 1=1;
```

Recursive common table expression generates all key values

A second common table expression sets the values for stale or fresh data

A Cartesian join combines all the values, and the case expressions run the data transformation rules

- This statement example generates data to populate a look-up table
  - Actual real statement generated 300,000 rows of data for a load
  - Prior to V8 the data would have to be manually entered!

V8 - Recursion – Generating Statements

- Two Queries to Generate 50,000 Statements
  - One sequential

```
WITH GENDATA (ACCT_ID, HIST_EFF_DTE) AS
(VALUES (CAST(1 AS DEC(11,0)), CAST('2004-02-01' AS DATE))
UNION ALL
SELECT ACCT_ID + 5, HIST_EFF_DTE
FROM   GENDATA
WHERE  ACCT_ID < 249996)
SELECT 'INSERT INTO YLA.ACCT_HIST (ACCT_ID, HIST_EFF_DTE)' CONCAT
' VALUES(' CONCAT CHAR(ACCT_ID) CONCAT ',' CONCAT ''''
CONCAT CHAR(HIST_EFF_DTE,ISO) CONCAT ''''  CONCAT ');'
 FROM GENDATA ORDER BY ACCT_ID;
```

50,000 inserts in key sequence, incrementing the ACCT_ID from 1 by 5 with each statement

  - One random

```
WITH GENDATA (ACCT_ID, HIST_EFF_DTE, ORDERVAL) AS
(VALUES (CAST(2 AS DEC(11,0)), CAST('2003-02-01' AS DATE), CAST(1 AS FLOAT))
UNION ALL
SELECT ACCT_ID + 5, HIST_EFF_DTE, RAND()
FROM   GENDATA
WHERE  ACCT_ID < 249997)
SELECT 'INSERT INTO YLA.ACCT_HIST (ACCT_ID, HIST_EFF_DTE)' CONCAT
' VALUES(' CONCAT CHAR(ACCT_ID) CONCAT ',' CONCAT ''''
CONCAT CHAR(HIST_EFF_DTE,ISO) CONCAT ''''  CONCAT ');'
 FROM GENDATA ORDER BY ORDERVAL;
```

50,000 inserts in random order (by using a RAND function)

NULL Evaluation Prior to V8

- Problem with coding for NULL values
- Nulls represent the absence of a value
- If a column has a NULL value we must carefully code to ensure we do not miss data that should be returned
  - Comparison can result in TRUE, FALSE, or UNKNOWN

COLA allows NULL values

Where cola = :var1 or (cola is null and :var1 is null)

Without the added null comparison, values may be missed because result will equate to unknown (I.e. not true)

Where t1.cola = t2.cola or (t1.cola is null and t2.cola is null)

Without the added null comparison, values may be missed in the join

V8 - NULL Evaluation …New Clause

- TRUE, FALSE or UNKNOWN
  - For SQL comparison results

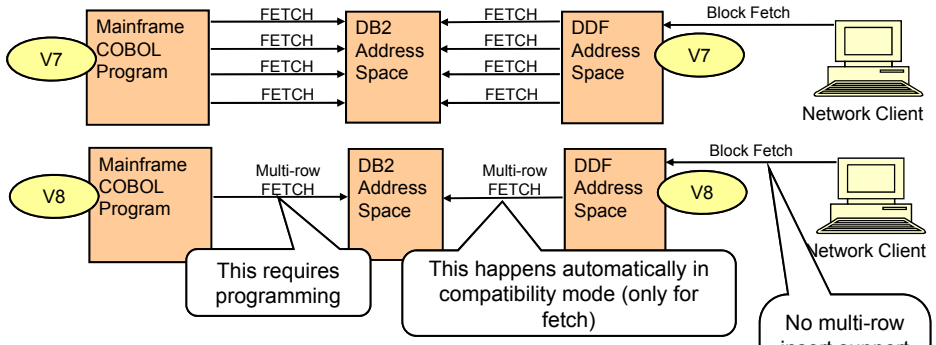
IS NOT DISTINCT FROM

- When comparing NULL values there is a new option
  - IS NOT DISTINCT FROM is the equals condition (“=“)
  - IS DISTINCT FROM is the not equals condition (“<>”)
  - Don’t understand? Substitute the word “different” for “DISTINCT”
    - IS NOT *‘different’* FROM
- Will result in a TRUE condition instead of UNKNOWN

```
SELECT COLA FROM TABLEA
WHERE COLA IS NOT DISTINCT FROM :var1
```

```
If COLA = NULL and :VAR1 = ‘YLA’   then FALSE
If COLA = NULL and :VAR1 = NULL    then TRUE
If COLA = ‘YLA’  and :VAR1 = ‘YLA’   then TRUE
If COLA = ‘YLA’  and :VAR1 = NULL    then FALSE
If COLA = ‘YLA’  and :VAR1 = ‘INC’   then FALSE
```


V8 - Multi-Row Fetch



- Multi-row fetch is specifically designed to save CPU
  - It is enabled for local application via specific SQL syntax
  - It is enabled for remote applications automatically
    - For block fetching cursors
- Going across address spaces is expensive
  - If we get multiple rows in one operation we can save CPU

V8 - Multi-Row Fetch

- Ability to return multiple rows on a single API call
  - Potential performance improvement up to 50%
- Static or dynamic
- Scrollable or non-scrollable cursor support
- Wide Cursors
  - Multiple rows, instead of one
- Supports positioned UPDATE and DELETEs
  - But there are caveats
- Watch out for locks!!
  - Every row being processed by a multi-row fetch must be locked
  - Cursor is position on all rows in current rowset
    - Locks will depend on isolation level and whether or not a result table is materialized
- Can mix single row and multi-row fetch in same cursor
  - FETCH NEXT is relative to first row in current rowset – Be careful!

Multi-Row Fetch Syntax

```
DECLARE CUR1 CURSOR
    WITH ROWSET POSITIONING
    FOR SELECT COL1, COL2 FROM TABLE1;
```

```
OPEN CUR1;
```

GET DIAGNOSTICS statement can return individual row operation execution information

```
FETCH NEXT ROWSET FROM CUR1
   FOR  :hv ROWS INTO :values1, :values2;
```

Fetch next rowset and position in rowset

Determines rowset size

If not specified for a cursor with ROWSET POSITIONING then the size of the rowset is the same as the previous rowset fetch

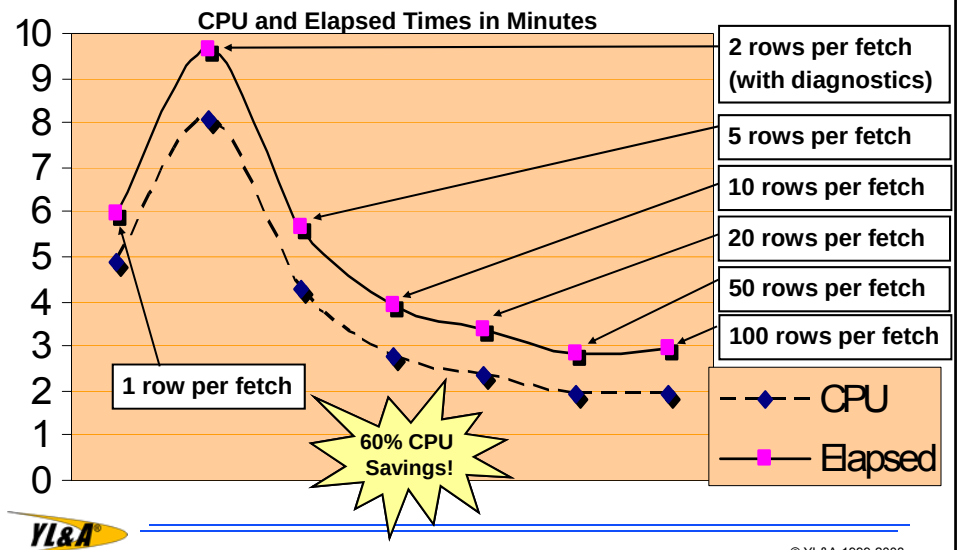
** You can adjust the rowset size…*

```
FETCH ROWSET FROM CUR1
    STARTING AT ABSOLUTE 10 FROM CUR
FOR 6 ROWS INTO :values1, :values2;
```

Fetch 6 rows starting from the 10th position in the rowset

Multi-Row Fetch - Impact

Local Cobol Test
Sequential Read of Approximately 39 Million Rows, 5 Columns



Multi-Row Fetch Test V8

Local Cobol Test
50,000 Random Read Requests at about 20 Rows Each

- Application is very dependent upon I/O response in a random application
  - No elapsed time improvement should be expected
  - Should get CPU improvement
- Actual result
  - One row fetch
    - 592 seconds elapsed, 22.5 seconds CPU
  - Twenty row fetch
    - 626 seconds elapsed, 16.7 seconds CPU
  - 25% CPU savings!

Remote Application

- Remote clients
  - SQLJ client experienced no CPU increase when moved from V7 to V8. Multi-row fetch offset additional V8 CPU cost

Coding Multi-row Fetch – Some Considerations

- There are a few caveats to remember when coding these cursors
  - Remember that once you start working with a rowset you should remain working with rowsets, otherwise things can get confusing
  - Do not mix multi-row fetches with single-row fetches in the same cursor
- The following example shows the results of doing a mix of multi-row (rowset) fetching, single rowset fetching, and single fetch.

This first fetch is for a rowset of 10.

```
FETCH COUNT IS 0000000010
IDS FETCHED:
CUSID IS 00000000262  ACCT IS 00000002537
CUSID IS 00000000262  ACCT IS 00000002538
CUSID IS 00000000281  ACCT IS 00000002672
CUSID IS 00000000281  ACCT IS 00000002673
CUSID IS 00000000372  ACCT IS 00000003503
CUSID IS 00000000372  ACCT IS 00000003504
CUSID IS 00000000372  ACCT IS 00000003505
CUSID IS 00000000372  ACCT IS 00000003506
CUSID IS 00000000372  ACCT IS 00000003507
CUSID IS 00000000372  ACCT IS 00000003508
```

**This next fetch will do a rowset fetch,
but for only one row.
It fetched a row from the next available rowset.**

```
FETCH NEXT ROWSET FOR 1 ROW
CUSID IS 00000000372  ACCT IS 00000003509
```


Coding Multi-row Fetch – Some Considerations

This next fetch will fetch another rowset of 10. Everything still looks good…

```
FETCH COUNT IS 0000000010
IDS FETCHED:
CUSID IS 00000000372  ACCT IS 00000003510
CUSID IS 00000000372  ACCT IS 00000003511
CUSID IS 00000000378  ACCT IS 00000003576
CUSID IS 00000000388  ACCT IS 00000003709
CUSID IS 00000000623  ACCT IS 00000006085
CUSID IS 00000000623  ACCT IS 00000006086
CUSID IS 00000000623  ACCT IS 00000006087
CUSID IS 00000000623  ACCT IS 00000006088
CUSID IS 00000000623  ACCT IS 00000006088
CUSID IS 00000000623  ACCT IS 00000006089
CUSID IS 00000000746  ACCT IS 00000007421
```

**Now the application decides to do a traditional single row fetch
(instead of a fetching a rowset of 1 as shown above)
The fetch grabs the second row of the previous rowset
due to the fact that it was positioned on the first row of the previously fetched rowset.
Is that what the application really wanted or did they want the next available row???**

```
FETCH 1 ROW NORMAL CURSOR
CUSID IS 00000000372  ACCT IS 00000003511 ←
```

Coding Multi-row Fetch – Some Considerations

Even more interesting is what happens when another rowset fetch of 10 is performed.
Since the cursor was positioned on ACCT 3511 in the previous rowset,
it now fetches the next rowset from that position.

```
FETCH COUNT IS 0000000010
IDS FETCHED:
CUSID IS 00000000378  ACCT IS 00000003576 ←
CUSID IS 00000000388  ACCT IS 00000003709
CUSID IS 00000000623  ACCT IS 00000006085
CUSID IS 00000000623  ACCT IS 00000006086
CUSID IS 00000000623  ACCT IS 00000006087
CUSID IS 00000000623  ACCT IS 00000006088
CUSID IS 00000000623  ACCT IS 00000006089
CUSID IS 00000000746  ACCT IS 00000007421
CUSID IS 00000000746  ACCT IS 00000007422
CUSID IS 00000000746  ACCT IS 00000007423
```

Bottom line: Do not mix cursor types!

V8 - Multi-Row Inserts

- **Ability to insert multiple rows with one API call**
  - **Potential 25% CPU reduction**
- **Static or Dynamic Support**
  - **For static, FOR 'n' ROWS on INSERT statement**
  - **For dynamic, FOR 'n' ROWS on EXECUTE statement**
- **Input values provided by host variable arrays**
- **Supports host language arrays**
  - **Assembler, C, C++, Cobol, PL/I**
- **Helps with application portability**
- **Great performance over single row inserts over a network**
  - **Rowset size becomes block size for query block**
  - **If 10 rows in rowset that is the block – 32K block size is turned off**
- **Insert up to 32767 rows per API call**

```
INSERT INTO TABLE1
    VALUES (:valuearray1,
                :valuearray2)
    FOR hv: ROWS
ATOMIC
```

V8 - Multi-Row INSERT (cont..)

- ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION
  - Whether or not all changes are done if an row to be inserted fails
- ATOMIC
  - All or nothing. The inserts are treated as a single statement
  - Careful with large amounts of data and rollback effects
  - Statement triggers are fired once
  - Rollbacks could be long and expensive
- NOT ATOMIC CONTINUE ON SQLEXCEPTION
  - Will continue if there are errors
  - Statement level triggers are fired once for each row
  - Diagnostics are available for each row that fails
    - Using GET DIAGNOSTICS
    - SQLCODEs all tell if all work(+252), if some did not(-253), or if all failed(-254)

V9 - MERGE Statement

- New statement combining conditional INSERT and UPDATE operations on a target using values from a source
- Given a set of input rows
  - Update the target table if the key exist
  - Insert the row for keys that do not exist
- Previously may have been accomplished with a conditional INSERT or UPDATE
- Good for OLTP applications

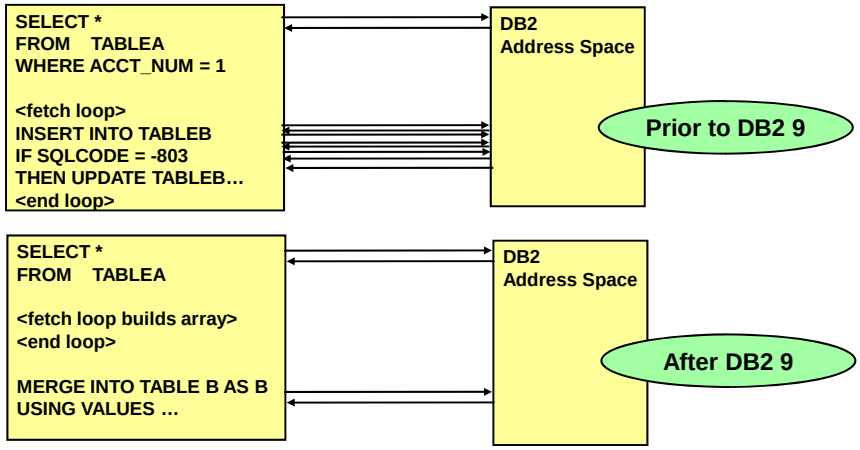
V9 - MERGE Statement

- This statement will merge the update values with the values in the ACCOUNT tables
- If the ACCT_ID exists the updates to the ACCT_BAL will occur, if not there will be an insert of the new ACCT_ID and ACCT_BAL

```
MERGE INTO ACCOUNT AS A
USING VALUES (:HV-ACCID, :HV-AMNT)
FOR 4 ROWS AS B(ACCT_ID, PAY_AMT)
ON A.ACCT_ID = B.ACCT_ID
WHEN MATCHED THEN
UPDATE SET ACCT_BAL = A.ACCT_BAL + B.PAY_AMT
WHEN NOT MATCHED THEN
INSERT (ACCT_ID, ACCT_BAL) VALUES (B.ACCT_ID, B.PAY_AMT)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```


V9 - MERGE for Data Replication

- In many situations applications are written to replicate data from one database to another
  - Use of MERGE can dramatically reduce the quantity of SQL statements required to do this!



V9 - INSTEAD OF Trigger

- An *instead of trigger* fires instead of the INSERT, UPDATE, or DELETE statement that activates the trigger
- Unlike the other triggers, this trigger type can be defined only against a view
- Used to insert, update, and delete data in complex views
  - Views defined on expressions or multiple tables
    - Such views are often read-only; in these cases, an INSTEAD OF TRIGGER can make insert, update, and delete operations against a read-only view possible

Read-Only View

```
CREATE VIEW EMPV
(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
PHONENO, HIREDATE, DEPTNAME) AS
SELECT EMPNO, FIRSTNME, MIDINIT,
LASTNAME,
      PHONENO, HIREDATE, DEPTNAME
FROM   EMP A
INNER JOIN
      DEPT B
ON     A.WORKDEPT = B.DEPTNO;
```

INSTEAD OF Trigger

```
CREATE TRIGGER EMPV_DELETE
INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP
FOR EACH ROW MODE DB2SQL
DELETE
FROM   EMP E
WHERE  E.EMPNO = OLDEMP.EMPNO;
```


V8 - Multiple DISTINCT Clauses

- Support for more than one DISTINCT keyword in a SQL statement
  - On SELECT or HAVING clause
- Performs multiple sorts on multiple distinct columns
- Can be costly depending on number of sorts and workfiles required
- Used to result in a SQLCODE –127

```
SELECT COUNT (DISTINCT col1), SUM (DISTINCT col2)
```

```
SELECT COUNT (DISTINCT col1), COUNT (ALL col1)
    FROM table1
    WHERE col2 > 1
    HAVING AVG (DISTINCT col3) > 0
```


44

Overuse of DISTINCT….

- Is the use of the DISTINCT clause justified in your query? Why is it there several times?
- Many code generators create DISTINCT after every SELECT Clause
  - No matter where the SELECT is
  - Some programmers code them as a safeguard
- The use of DISTINCT can cause excessive sorting
  - DISTINCT in a table expression forces materialization
  - Only a unique index can avoid sorting for DISTINCT
- Are Duplicates Possible?
  - No, then remove the DISTINCT!
  - If so, can a sort be avoided
    - Unique index can be used to avoid a sort
    - GROUP BY all columns can use non-unique indexes
  - Use DISTINCT as early as possible in complex queries
    - Get rid of the duplicates as early as possible
    - Avoid using DISTINCT more than once in a query

V9 – DISTINCT Sort Improvement

- V8 Challenge - DISTINCT can only use a unique index to avoid sort for DISTINCT
- V9 - DISTINCT query can have sort avoidance via a non-unique index

```
SELECT DISTINCT ACCT_NO FROM ACCOUNT
```

Non-unique Index on ACCT_NO

- Using non-unique index on ACCT_CD:

```
SELECT DISTINCT(ACCT_CD) FROM ACCOUNT
ORDER BY ACCT_CD
FETCH FIRST 10 ROWS ONLY;
```

Non-unique Index on ACCT_CD

- V8 - a non-matching index only scan via a non-unique index, with a sort
- V9 - a non-matching index only scan via a non-unique index with no sort
- V9 - Workfile allocation is avoided for the FETCH FIRST clause if the result can fit within a single page
- V9- CPU and getpages savings for queries containing GROUP BY or DISTINCT

Optimistic Locking Challenge in V8

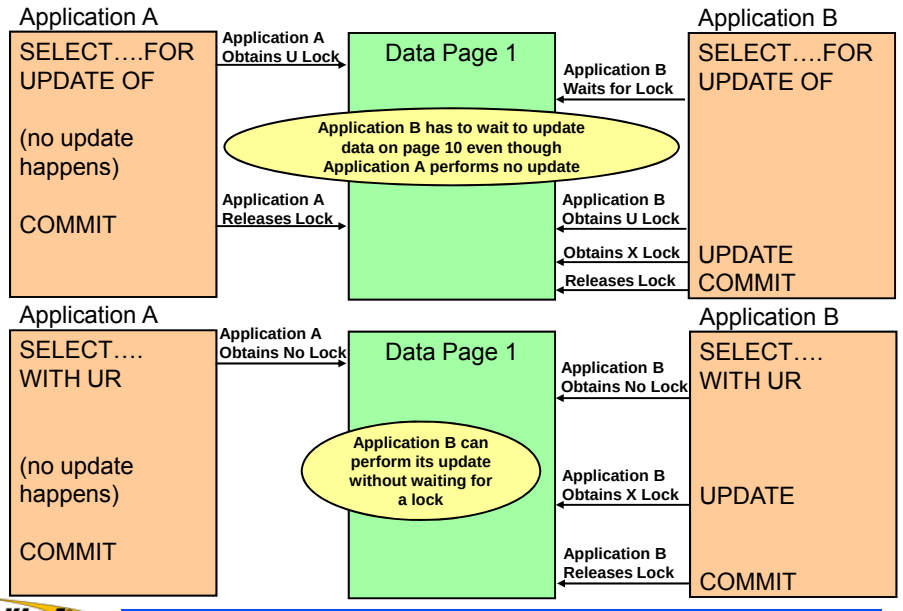
- Use optimistic locking to insure update integrity
  - All tables have an update timestamp column
  - Read data WITH UR
  - Include the timestamp in any UPDATE or DELETE

```
SELECT PKG_ID, PKG_DESC, UPD_TSP
FROM PKG_TBL
WHERE PKG_ID = :PKG-ID WITH UR
```

Optimistic locking uses the update timestamp to insure no other process has modified the data, and is built into DB2 for z/OS V9

```
UPDATE PKG_TBL
SET PKG_DESC = :PKG-DESC
    ,UPD_TSP = CURRENT TIMESTAMP
WHERE  PKG_ID    = :PKG_ID
AND    UPD_TSP = :UPD-TSP
```


Optimistic Locking in Action



Optimistic Locking in V9

- Optimistic Locking (a.k.a concurrency control)
  - Faster more scalable locking alternative to database locking for concurrent access
  - Used to minimize the time for which a given resource is unavailable for use by other transactions
  - Locks for Reads
    - Obtained immediately before a read operation and released
  - Locks for Updates
    - Obtained immediately before an update operation and held until the end of the transaction
- Optimistic locking test whether the underlying data source has changed by another transaction since the last read operation
  - Prior to optimistic locking, all columns marked for update and their values in last read operation are added explicitly through a WHERE clause in the UPDATE statement
    - Therefore the UPDATE fails if the underlying column values have been changed

Optimistic Locking Programming in V9

- It is optimistic locking support
  - Not automatic optimistic locking
  - You still have to program for it
- Advantage is that DB2 is in control
  - At page level if no ROW CHANGE TIMESTAMP column
  - At row level if there is a ROW CHANGE TIMESTAMP column

```
SELECT PKG_ID, PKG_DESC,
          ROW CHANGE TIMESTAMP FOR PKG_TBL
FROM PKG_TBL
WHERE PKG_ID = :PKG-ID WITH UR
```

```
UPDATE PKG_TBL
SET PKG_DESC = :PKG-DESC
WHERE  PKG_ID    = :PKG_ID
AND      ROW CHANGE TIMESTAMP FOR PKG_TBL = :UPD-TSP
```

Skip Uncommitted Insert – V8

SKIPUNCI

- **Specifies whether statements ignore a row that was inserted by a transaction (other than itself)**
  - **And that has not yet been committed or aborted**
- **Applies only to statements running with row-level locking and isolation-level read stability or cursor stability**
- **Applies to ISO (CS or RS)**
  - **Yes – skip uncommitted insert**
    - **DB2 behaves as though the uncommitted row has not yet arrived and the row is skipped**
  - **No(default) – DB2 waits for the inserted row to be committed or rolled back**
    - **Then processes the row as a qualifying row if the insert commits or moves on to find another row if the insert is rolled back**
    - **If a transaction performs one or more inserts, then spawns a second transaction, specify NO for SKIP UNCOMM INSERTS if the first transaction needs the second transaction to wait for the outcome of the inserts**
- **YES offers greater concurrency than the default value of NO**

Skipped Locked Rows in V9

- SKIPPED LOCKED DATA option
  - Allows a transaction to skip rows that are incompatibly locked by other transactions
- Can help to improve performance of some applications
  - By eliminating lock wait time
- Should only be used for application that to not require unavailable or uncommitted data
- Transactions uses this option will not read or modify data that is unavailable, uncommitted or held by locks
- SKIP LOCKED DATA options can be specified in
  - SELECT, SELECT INTO, PREPARE, searched UPDATE, or searched DELETE statement
- Can also be used with the UNLOAD utility
- Works with isolation level CS or RS
  - Will be ignored for UR or RR

Will return a count of only the rows that are not uncommitted
(if no index on CUST_NO)

```
SELECT COUNT(*)
FROM CUSTOMER
WHERE CUST_NO >= 500
SKIP LOCKED DATA
```

ORDER BY and FETCH FIRST in Subselect or Fullselect – V9

- Support for ORDER BY or FETCH FIRST n ROWS in subselect or fullselect
- In V9, all semantically correct clauses of a SELECT statement can also be put in a subselect or fullselect
- Also provides more function
  - Can select top 'n' rows from
    - Result table of a table expression
    - Leg of a Union
    - Subquery
- ORDER BY usually does not have bearing on result table
    - If FETCH FIRST is used with ORDER BY

ORDER BY and FETCH First in Subselect or Fullselect – V9

- This is an exciting and powerful feature
  - It allows for all sorts of powerful existence checking

List the balance and most recent status for account 'A000010'

```
SELECT  A.ACCT_BAL, H.ACCT_STATUS
FROM    ACCOUNT A, ACCT_STS_HIST H
WHERE   A.ACCT_ID = H.ACCT_ID
AND     A.STATUS_DTE =
        (SELECT MAX(STATUS_DTE)
          FROM    ACCT_STS_HIST X
          WHERE   X.ACCT_ID = H.ACCT_ID)
AND     A.ACCT_ID = 'A000010';
```

Status is stored in a history table

Which will be the performance winner?

```
SELECT  A.ACCT_BAL, H.ACCT_STATUS
FROM    ACCOUNT A,
TABLE   (SELECT ACCT_STATUS
          FROM ACCT_STS_HIST X
          WHERE  A.ACCT_ID = X.ACCT_ID
          ORDER BY STATUS_DTE DESC
          FETCH FIRST 1 ROW ONLY) AS H
WHERE   A.ACCT_ID = 'A000010';
```

FETCH FIRST and ORDER BY gives the same result with one probe of ACCT_STS_HIST

Scalar and Built-in Functions – V9

SCALAR

ASCII_CHR
ASCII_STR
COLLATION_KEY
DECRYPT_BINARY
DIFFERENCE
EBCDIC_CHR
EBCDIC_STR
EXTRACT
LOCATE_IN_STRING
LPAD
MONTHS_BETWEEN
NORMALIZE_STRING
OVERLAY
RID
RPAD
SOUNDEX
TIMESTAMPADD
TIMESTAMPDIFF
TIMESTAMP_FORMAT

- There are several new functions in DB2 9

SCALAR (cont…)

TIMESTAMP_ISO
UNICODE
UNICODE_STR
VARCHAR_FORMAT

AGGREGATE

CORRELATION
COVARIANCE
COVARIANCE_SAMP

Many Cool New Functions – Native SOUNDEX – V9

- No soundex function prior to DB2 9
- We had to code it ourselves
- This came close

Soundex aids in name search operations. Now built-in!

Before DB2 9

```
--
-- SQL SCALAR UDF FOR RUSSEL SOUNDEX
-- WARNING: ACCURACY NOT GUARANTEED, USE AT OWN RISK
--
CREATE FUNCTION YLA.SOUNDEX(SNME VARCHAR(255))
RETURNS CHAR(4)
LANGUAGE SQL
CONTAINS SQL
NO EXTERNAL ACTION
DETERMINISTIC
RETURN
CONCAT(SUBSTR(LTRIM(REPLACE(SNME,'','')),1,1),REPLACE(SUBSTR(LTRIM(REPLACE(TRANSLATE(REPLACE
(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(TRANSLATE(UPPER(LTRIM(REPLACE(SNME,'','')))
,'123122245512623122','BCDFGJKLMNPQRSTVXZ'),' ',''),'-',''),'11','1'),'22','2'),'33','3'),'44','4'),'55','5'),'66','6')
,'11','1'),'22','2'),'33','3'),'44','4'),'55','5'),'66','6'),'11','1'),'22','2'),'33','3'),'44','4'),'55','5'),'66','6')
,'11','1'),'22','2'),'33','3'),'44','4'),'55','5'),'66','6'),'','AEIOUYHW'),' ',')),CASE SUBSTR(LTRIM(REPLACE(SNME,'','')),1,1)
WHEN 'A' THEN 1 WHEN 'E' THEN 1 WHEN 'I' THEN 1 WHEN 'O' THEN 1 WHEN 'U' THEN 1 WHEN 'Y' THEN 1 WHEN
'W' THEN 1 WHEN 'H' THEN 1 ELSE 2 END,3),' ','0'));
```

```
SELECT YLA.SOUNDEX('LAWSON')
FROM SYSIBM.SYSDUMMY1
```

Before DB2 9

L250

```
SELECT SOUNDEX('LAWSON')
FROM SYSIBM.SYSDUMMY1
```

After DB2 9

Many Cool New Functions – Native MONTHS_BETWEEN

- No such function prior to DB2 9
- We had to code it ourselves

```
CREATE FUNCTION TOTMON (STARTX DATE, ENDY DATE )
        RETURNS INTEGER
        LANGUAGE SQL
        CONTAINS SQL
        NO EXTERNAL ACTION
        NOT DETERMINISTIC
        RETURN ABS( (YEAR( STARTX – ENDY )*12 ) + MONTH( STARTX – ENDY ) );
```

Before DB2 9

```
SELECT YLA.TOTMON(CAST('2007-10-11' AS DATE), CAST('2007-11-11' AS DATE))
FROM SYSIBM.SYSDUMMY1

1
--
1
```

```
SELECT MONTHS_BETWEEN('2007-10-11','2007-11-11')
FROM SYSIBM.SYSDUMMY1

1
-----------------------------
1.000000000000000
```

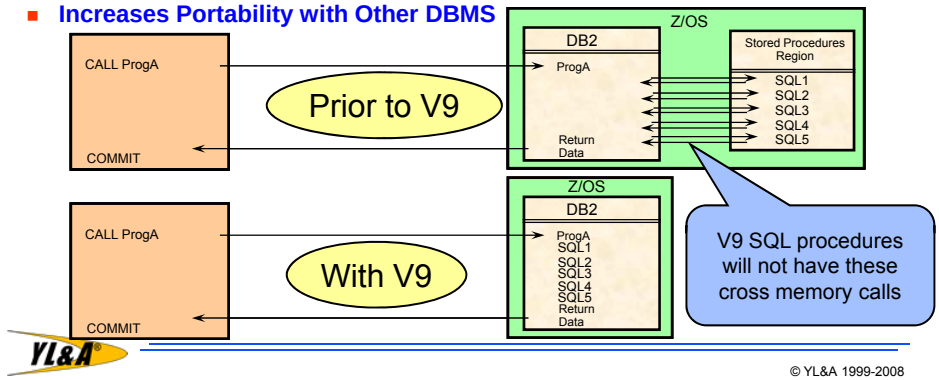
After DB2 9

SQL Procedure Support – Native SQL Procedures – V9

- A SQL procedure is written entirely of SQL and control statements
  - External procedures are written in other languages
- Prior to V9
  - SQL procedures were converted to C programs and were subject to same considerations as other external procedures
- DB2 V9
  - Does not require a C compiler
    - When created, the statements are converted into a native representations
      - Stored in directory (like other SQL statements)
      - Parameter list and options are stored in catalog
    - When called the native representation is loaded and executed by DB2 engine
      - Will have performance advantages
        - Execution done in a single API call
    - zIIP enabled

Native SQL Procedure Support – V9

- SQL Stored Procedures Prior to V9
  - SQL procedures ultimately become C programs
  - Procedures execute in WLM managed stored procedure address space
- Native SQL Procedure Language Support
  - Does not require a C compiler
  - Instead of a C program you get a runtime DB2 structure
    - Dramatic performance improvement since no cross-memory call to the stored procedure address space
- Increases Portability with Other DBMS



Optimization
Challenges and V8/V9 Solutions

Distribution Statistics – Challenge – V8/V9

- Many performance problems in the past have been due to skewed distribution of data
  - Can result in
    - Poor join sequences
    - Increase in synchronous I/O
    - Increase application response times
  - This can occur because the optimizer could choose an incorrect access path because of an incorrect estimation of the number of matching row for a qualified predicate
  - DB2 maintains distribution in the catalog table SYSIBM.SYSCOLDIST
    - If used with cardinality statistics in SYSIBM.SYSCOLUMNS performance may improve
    - This optimization effort can be expanded further by holding frequency and cardinality statistics not just for single columns but for skewed combinations of columns
- Need distribution statistics for
  - Non-leading indexed column or non–indexed columns
- Give DB2 more information for better optimization

Non-Index and Non-Uniform Distribution Statistics – V8

- COLGROUP
  - Set of columns representing a group
  - Cardinality values will be collected on the group
  - Allows for collection of distribution statistics for non-indexed or non-leading indexed columns
  - Specify FREQVAL also to collect frequency statistics
- MOST
  - Collects the most frequently occurring values for the set of columns
  - Collects the correlation statistics for most frequently occurring values in the column group
- LEAST
  - Collects the least frequently occurring values for the set of columns
  - Collects the correlation statistics for least most frequently occurring values in the column group
- BOTH
  - Collects the least/most frequently occurring values for the set of columns
  - Collects the correlation statistics for least least/most frequently occurring values in the column group

```
COLGROUP (c1,c2) FREQVAL COUNT 10 BOTH
```

When to Use Frequency/Distribution Statistics

- **Problem: DB2 assumes data is uniformly distributed and all values in the column occur with the same frequency**
  - **Inaccurate estimates on qualifying rows**

DB2 Assumes – 1/COLCARDF – 20%

DB2_Skills	*Frequency*
Guru	*5%*
Senior	*20%*
Junior	*40%*
Trainee	*10%*
Needs-new-job	*25%*

WHERE DB2_SKILLS = 'Junior'

DB2 estimates – 20%, but this is 50% underestimated since the actual percentage is 40%

```
SELECT DB2_SKILLS, COUNT(*)
 FROM IT_EMP
 GROUP BY DB2_SKILLS
ORDER BY 2 DESC
```

Query to find Data distribution

DB2_Skills	
Guru	5
Trainee	8
Senior	20
Needs-new-job	24
Junior	55

For static values these statistics may not have to be collected more than once.

Query Performance without Distribution Statistics

- **This Query Suffers From Poor Performance**

```
SELECT *
 FROM  PDB2.CONTACTNOTICE AS CN
  INNER JOIN TDB2.KEY_NOTICE AS KN
   ON  CN.COD_CLIENT       =KN.COD_CLIENT
   AND CN.COD_GENERATION   =KN.COD_GENERATION
   AND CN.ID_CONTACTNOTICE=KN.ID_CONTACTNOTICE
  INNER JOIN PDB2.CUST AS CU
   ON  KN.COD_CLIENT       =CU.COD_CLIENT
   AND KN.COD_GENERATION   =CU.COD_GENERATION
   AND KN.ID_CUST_JOB      =CU.ID_CUST
  INNER JOIN PDB2.CUST AS CU2
   ON  KN.COD_CLIENT       =CU2.COD_CLIENT
   AND KN.COD_GENERATION   =CU2.COD_GENERATION
   AND KN.ID_CUST_1        =CU2.ID_CUST
  INNER JOIN TDB2.KEY_PERSON AS KP
   ON  CU2.COD_CLIENT      =KP.COD_CLIENT
   AND CU2.COD_GENERATION =KP.COD_GENERATION
   AND CU2.ID_CUST         =KP.ID_CUST
  WHERE CN.DOM_NOTIFY          = 'FICH'
    AND CN.COD_GENERATION      = 'MB'
    AND CN.COD_CLIENT          = '0450'
    AND CN.DOM_STATUS_NOTICE   = 'ERL'
    AND CU2.DOM_PERSONGROUP    = 'PR'
    AND CU2.COD_LAND_DIVISION = 'AT'
    AND CU2.DOM_STATUS         = 'BEST'
    AND CU2.DOM_CARE_STATE     = 'S';
```

Query Response: 20 min.
Expected Response: <1 min.

DB2 has used column statistics to compute filter factors for predicates, and determined that table CU2 will return only 1 row, and has picked that table first in the join sequence.

Optimizer Estimates

Table	Estimate	% of cardf
CUST (CU)	58,039	0.2%
CUST (CU2)	1	0.000003%
CONTACTNOTICE	704	0.0009%
KEY_PERSON	19,619	0.03%
KEY_NOTICE	156,163	0.65%

Query Performance without Distribution Statistic

- We Can Compare Actual Row Counts to the Optimizer Estimate
  - The Data is Skewed, but the Current Statistics Don't Reflect That
  - Example of a Count Query for Table CU2:

```
SELECT COUNT(*) = 267,011
FROM  PDB2.CUST AS CU2
WHERE CU2.DOM_PERSONGROUP   = 'PR'
  AND CU2.COD_LAND_DIVISION = 'AT'
  AND CU2.DOM_STATUS        = 'BEST'
  AND CU2.DOM_CARE_STATE    = 'S'
  AND CU2.COD_CLIENT        = '0450'
  AND CU2.COD_GENERATION    = 'MB'
```

- Comparison of all Counts and Estimates:

	From Count Queries		Optimizer Estimate	
Table	Count	% of cardf	Estimate	% of cardf
CUST (CU)	420,973	1.45%	58,039	0.2%
CUST (CU2)	267,011	0.92%	1	0.000003%
CONTACTNOTICE	1,472	0.002%	704	0.0009%
KEY_PERSON	20,114	0.03%	19,619	0.03%
KEY_NOTICE	156,347	0.65%	156,163	0.65%

Distribution Statistics in Action – V8

- RUNSTATS is Used to Collect Distribution Statistics
  - Statistics Used for Dynamic Queries with Embedded Literals
  - Results in Filter Factor That More Accurately Reflects Reality

```
RUNSTATS TABLESPACE IDVKUNDA.IDVCUST
      TABLE(PDB2.CUST)
      COLGROUP(COD_CLIENT)              FREQVAL COUNT 10
      COLGROUP(DOM_STATUS)              FREQVAL COUNT 10
      COLGROUP(COD_LAND_DIVISION)  FREQVAL COUNT 10
      COLGROUP(DOM_PERSONGROUP)  FREQVAL COUNT 10
      COLGROUP(DOM_CARE_STATE)     FREQVAL COUNT 10
```

CONACTNOTICE now first table accessed. Query response improved from 20 min to 20 sec

- Collecting Distribution Statistics on all Tables Results in a new Table Access Sequence

	From Count Query		New Optimizer Estimate	
Table	Count	% of cardf	Estimate	% of cardf
CUST (CU)	420,973	1.45%	419,204	1.45%
CUST (CU2)	267,011	0.92%	240,891	0.91%
CONTACTNOTICE	1,472	0.002%	704	0.0009%
KEY_PERSON	20,114	0.03%	19,619	0.03%
KEY_NOTICE	156,347	0.65%	156,163	0.65%

V8 SQL Distribution Statistics Challenge

- Distribution statistics are only gathered for the most or least *n* occurring values
- Suppose we had gathered the top 5 values for a column such as this chart shows

Table CARDF is 10,865,917, column COLCARDF is 457

TBNAME	NAME	COLVALUE	CARDF
TABLE1	COL1	AT	1,250,807
TABLE1	COL1	RP	400,211
TABLE1	COL1	GH	375908
TABLE1	COL1	FF	303,515
TABLE1	COL1	XZ	293,011

DB2 knows that about 1,250,807 rows will be returned from this query

```
SELECT *
FROM TABLE1
WHERE COL1 = 'AT'
```

For this query DB2 assumes 18,236 rows

```
SELECT *
FROM TABLE1
WHERE COL1 = 'YA'
```

DB2 has to assume a uniform distribution of the rest of the values

((1-(2623452/10865917))/(457-5)) * 10865917

V9 Histogram Statistics and Exploitation

- New type of data distribution statistics
- Used to enhance predicate selectivity estimation for better access paths
- Histogram statistics provides a way of summarizing data distribution on interval scale
  - Either discrete of continuous
  - Divides up the range of possible values in a dataset into quantiles
    - For which a set of statistics parameters are collected
- Based on equal-depth histogram statistics
  - Cuts the whole value range so that each quantile has about the same number of rows
    - Total number of quantiles
      - For each quantile
        - The pair of LOWVALUE/HIGHVALUE
        - Number of distinctive values (CARD)
        - Frequency (number of rows)
- Multi-column Histogram statistics are gathered the same as multi-column frequency statistics
  - Concatenated multiple columns and treat as a single value

V9 Histogram Statistics

- Histogram statistics are gathered for each quantile to a maximum of 100 quantiles
- This covers the entire range of values via RUNSTATS

COLGROUP(COL1) HISTOGRAM NUMQUANTILES 5

Table CARDF is 10,865,917, column COLCARDF is 457

TBNAME	NAME	QUANTILENO	LOWVALUE	HIGHVALUE	CARDF	Freq
TABLE1	COL1	1	AA	AT	5	21%
TABLE1	COL1	2	AU	FF	118	19%
TABLE1	COL1	3	GA	GH	8	20%
TABLE1	COL1	4	GI	RP	212	20%
TABLE1	COL1	5	RQ	XZ	114	20%

DB2 assumes that no rows will be returned from this query

(1/(quantile cardf)) *
(10865917*(quantile freq%))

```
SELECT *
FROM TABLE1
WHERE COL1 = 'YA'
```

The entire range of values are represented

Assuming the COLCARF of the quantile and frequency % of the quantile
This is more accurate than with just the frequency distribution statistics

REOPT(ONCE) – V8

- For dynamic SQL normally the access path is calculated at PREPARE
- New Bind Option
  - Tries to combine benefits of REOPT(VARS) and dynamic statement caching
  - Controls when an access path is obtained (reoptimized) for a statement
    - For dynamic SQL only
    - Frequently running short SQL statements
    - Statements in a loop
    - Statements used by multiple threads
  - With REOPT(VARS), can be expensive and can effect overall application performance

Consider separating these statements into their own package and BIND with REOPT(ONCE)

- With new option REOPT(ONCE)
  - Access Path selection will be deferred until the OPEN
  - Host variable values will then be used to calculate the access path
  - The access path will then be cached in the global prepare cache
  - Will use same variables for next execution
    - Better than using default variables

Autonomic Reoptimization – V9

- **Reoptimization for queries - bind parameters**
  - Static Embedded SQL with host variables
    - REOPT(ALWAYS)
      - May want to consider if there is a distribution issues with values at execution that could influence the access paths of the queries
      - Will cause incremental binds to occur
  - Dynamic SQL
    - REOPT(ONCE)
      - Will reoptimize the query once at OPEN and then store the access path in the dynamic SQL cache and use from there

    REOPT(AUTO)

      - Automatic reoptimization of the dynamic statement when DB2 detects the filtering of one or more predicates has big change
      - Will be especially helpful for queries/predicates with skewed data values

Autonomic Reoptimization V9 (cont…)

- **New DSNZPARM**

  REOPTEXT

  - Options
    - NO – works like V8
    - YES – available for SMART/AUTO
- **AUTO**
  - For dynamic SQL with parameter markers
    - Automatic reoptimization when DB2 detects filtering of one or more predicates changed greatly
    - New access path is generated
      - Will replace the current one
      - Placed in statement caches
    - Will report at beginning
      - Then monitor runtime values for parameter markers
    - First time optimization is same as ONCE

Plan Stability – V9

- **Challenge in V8**
  - Falling back to previous access path after a rebind happens that results in a new bad access path
- **Backup of static SQL packages**
- **Default controlled via DSNZPARM**
- **Supported on REBIND**
  - Ability to save old copies of packages
    - Saved in catalog/directory
  - Can switch back to
    - Current/previous version
      - PLANMGMT(BASIC)
    - Original (EXTENDED)
      - PLANMGMT(EXTENDED)

Case Insensitive Comparison – V9

- **V8 Challenge: Upper case displays are not desirable**
  - For usability and ease-of-use
  - Preference is mixed case data display
  - Character data displayed and is part of a query predicate
  - To maximize query efficiency
    - Best to store data in upper case
  - Makes the searches easy and indexable
    - However, it is difficult to reformat the display into a mixed format
  - Searching on mixed case data is very problematic
    - A LIKE predicate and a scalar function are needed
    - However, access path are not optimal

Case Insensitive Comparison – V9 (cont..)

- V9
  - It possible to store data as mixed case
  - And run case insensitive searches
  - Does not require table scans to return the results
- Example:

> Illinois, ILLINOIS, and illinois are returned by a query
> No need for where upper(:hv) =upper(column)

Sequential Inserts in V8

- Design key for sequential inserts at end
- No Freespace or Pctfree on data or sequential index
  - With no freespace you can reduce number of index levels and pages
- Can eliminate read I/Os
- No data read time needed
- Efficient use of IPROCs
- Index Lookaside on Clustering Index
- Less writes needed (less pages to be written)
- Will utilize deferred writes
- Minimizes read/write I/O, Getpages, and Locking
- MEMBER CLUSTER Setting Impacts Performance
- Index Design Affects Performance
- PQ87381 Needs to be Applied
  - Improved insert algorithm

New inserts at end together, Very efficient deferred write

Date	Acctno
12/01/02	1234
12/02/02	1234
12/05/02	5893
12/25/02	7892
01/01/03	7892
01/20/03	0414

V9 - APPEND on Insert

- Challenges still remain in V8
  - Clustering may not be beneficial
  - Maybe requirements of MEMBER CLUSTER and zero freespace cannot happen
- V9 will allow for fast inserts at the end of the table or partition
  - At the expense of organization
  - May account for some faster table growth
  - Avoids overhead of attempting to preserve clustering sequence
  - Maximizes performance for Inserts at the end
- Clustering can still be achieved by a reorg
  - With an explicit clustering index defined

CREATE TABLE …APPEND YES/NO

ALTER TABLE…APPEND YES/NO

Some Pros and Cons of DPSIs – V8

- Some of the benefits that this provides include:
  - Clustering by a secondary partitioned index
  - Ability to rotate partitions easily
  - Allow for reducing overhead in data sharing(affinity routing)
  - Further partition independence
  - Many utility benefits (discussed later)

You can do this operations with NPSIs, but there is more overhead

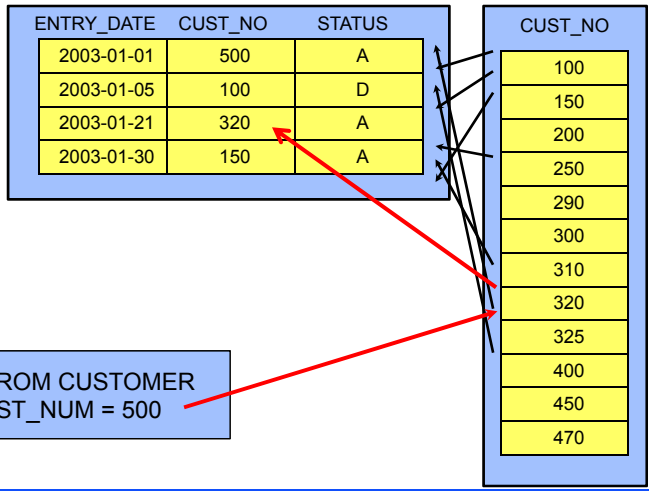
ENTRY_DATE	CUST_NO	STATUS
2003-01-01	500	A
2003-01-05	100	D
2003-01-21	600	A
2003-01-30	150	A
2003-02-02	300	A
2003-02-06	450	D
2003-02-22	200	A
2003-02-28	950	A
2003-03-01	800	A
2003-03-04	750	D
2003-03-06	320	A
2003-03-30	940	A

CUST_NO
100
150
500
600
200
300
450
950
320
750
800
940

Total Independence

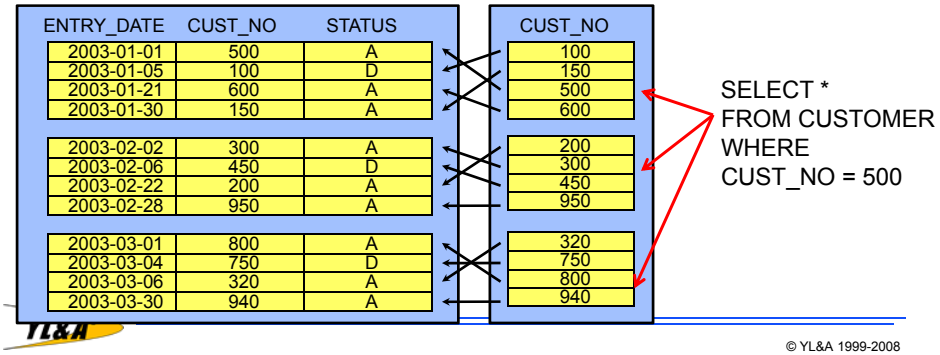
Coding Predicates for NPSIs – V8

- **Coding a predicate for an NPSI was easy**
  - **There WHERE clause identified the column in the NPSI**
  - **The NPSI is a 'single' object that points into all partitions of the tablespace**



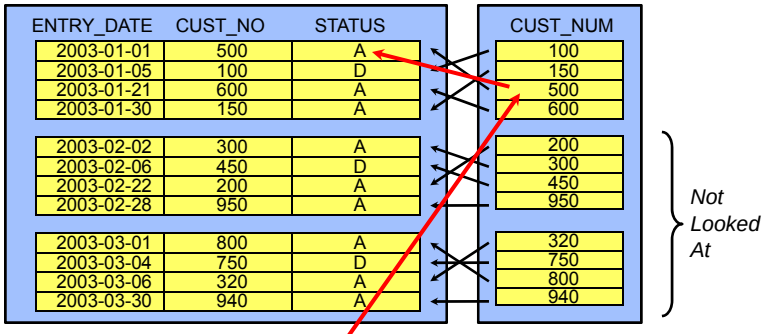
SQL Access Path Issues with DPSI – V8

- **Some queries may not perform as well**
- **If predicates that reference columns in a single partition and therefore are restricted to a single partition of the DPSI, it will benefit from this new organization**
  - **Queries have to be designed to allow for partition pruning through the predicates**
  - **If a predicate references only columns in the DPSI it may not perform well because it may need to probe several partitions of the index**



Predicates for DPSIs – V8

- Need to change predicate for efficient process against the DPSI
  - Code the partitioned index restriction to allow for partition pruning
    - Will need to know the correlation between DPSI and PI keys




```
SELECT * FROM CUSTOMER
WHERE CUST_NUM = 500
AND ENTRY_DATE
   BETWEEN '01-01-2003' and '01-30-2003'
```

This only works if we know that CUST_NUM 500 was issued between those dates

DPSI Queries – V8 Challenges

- Additional probes may be necessary to find data
- If DPSI is the inner table of a nested loop or hybrid join
  - All DPSI partitions will be accessed for every outer row
    - Significant increase in getpage count
      - By a factor of number of the inner table partitions per outer row
- DPSI and Parallelism
  - Does not work in V8
- Page range screening
  - DPSI predicates can act like index matching predicates
  - Can be used even without the index
    - Similar to index matching
    - Need predicate on limit key

DPSI on CUSTID
PARTITIONED BY ORDER_DATE

```
SELECT * FROM CUSTOMER
WHERE CUSTID = 4 AND
ORDER_DATE = '2005-01-01
```

Match on CUSTID (DPSI partition)
Match on ORDER_DATE (limit)

DPSI Queries…some more challenges…

DPSI on CUST_ORDER

DPSI on CO.CUSTID
PARTITIONED BY CO.ORDER_DATE

May also have issues with index lookaside and sequential detection resulting in more getpages

```
SELECT *
FROM CUSTOMER C, CUST_ORDER CO
WHERE C.CUSTID = CO.CUST_ID
AND ORDER_DATE = ?
```

Local Predicate on limit key
- Will have partition elimination

Partition elimination only works if …
1. *There is a local predicate (literal value, host variable, parameter marker, special register), on the leading partition limit key(s)*
2. *Or, a local predicate can be transitively closed against the leading partition limit key(s).*

```
SELECT *
FROM CUSTOMER C, CUST_ORDER CO
WHERE C.CUSTID = CO.CUST_ID
AND C.CUST_DATE = CO.ORDER_DATE
```

Join predicate on limit key
- No partition elimination

V9 – Join predicate on limit key can be used for partition elimination

Page Range Screening/Partition Elimination Improvements V9

- Improvements for partition elimination
- Join predicates

```
SELECT *
FROM CUSTOMER C, CUST_ORDER CO
WHERE C.CUSTID = CO.CUST_ID
AND C.CUST_DATE = CO.ORDER_DATE
```

Can now be used for partition elimination

- Non-matching predicates

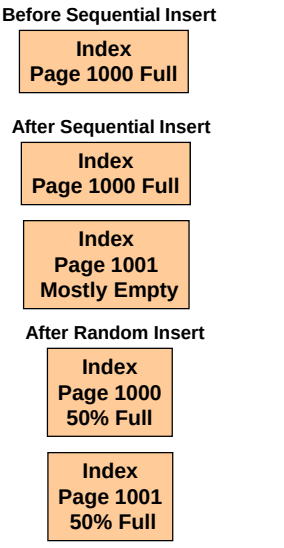
PARTITIONED BY ORDER_DATE AND ORDER_TYPE

```
SELECT * FROM CUSTOMER WHERE
ORDER_TYPE = 'DISCOUNT'
```

Can now be used for partition elimination even though ORDER_DATE is missing

Inserts and Large Indexes In V8 – Challenging with Splits!

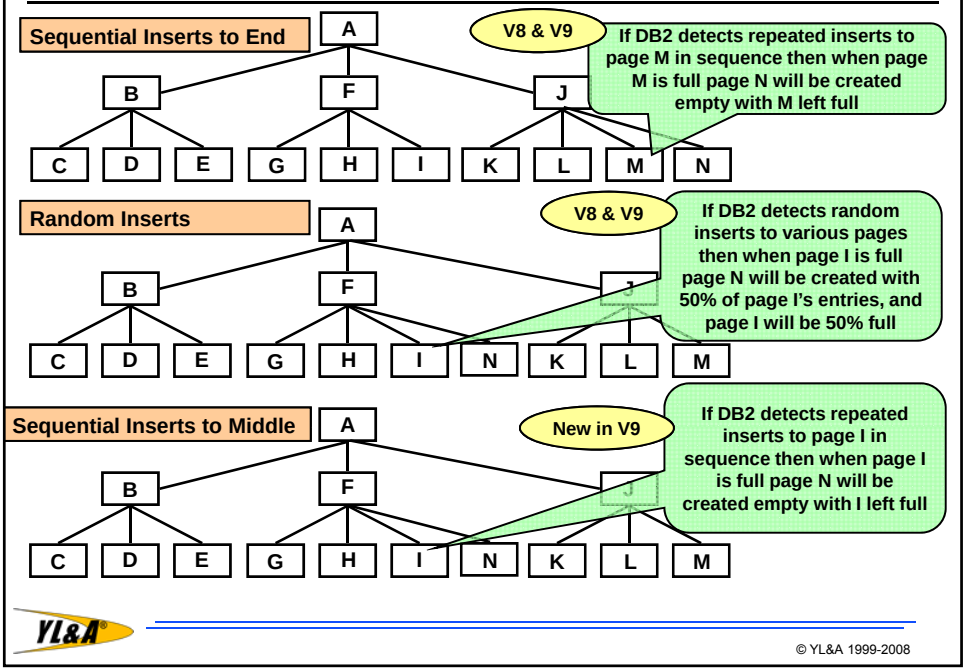
- If Purely Sequential Index Inserts
  - DB2 will not split pages at end
- If Not Purely Sequential Index Inserts
  - DB2 will split pages 50/50
  - Exhaustive search for available page
- Splits and Exhaustive Searches
  - DB2 will look for nearby page
  - No nearby page
    - Exhaustive search via spacemap chain
    - Huge cost!
- Bottom line on indexes and inserts
  - Pure sequential
    - No pctfree or freepage
  - Random
    - Set freepage
    - Better yet, large pctfree
      - So you never ever split!



Improved Index Page Splits in V9

- Sequential inserts can cause index splits leaving 50% of the page empty and not immediately used
  - Caused by inserts into the middle of the index
  - With 4K index pages this can start to cause more frequent splits and increased contention on the index tree
- In V9 there will be a asymmetric split of the index pages
  - Will better accommodate insert patterns into the index
  - Will allow for better space allocation
- Larger than 4K page sizes are now allowed for indexes
  - Larger number of index keys can be held on a page
  - Will help to reduce the amount of splitting necessary
    - Relieving contention on the index pages
- DB2 will detect insert pattern
  - Will select from different index page split algorithms
    - If random, DB2 will do a 50/50 page split
    - If sequential, DB2 will do an asymmetric split
      - Existing keys will move to new page to make room for new inserts

Splitting Index Pages in V9



Additional Index Page Sizes – V9

- Prior to V9
  - Size of an index page is limited to 4 KB
- Size of an index page limits the number of index keys that the index page can accommodate
  - Can cause contention in indexes that split frequently
- In V9
  - Offers expanded index page sizes of 8 KB, 16 KB, and 32 KB
- An index page size greater than 4 KB accommodates more index keys per page
  - Can reduce the frequency of index page splits for long keys
- The larger page size also helps with the compression of indexes

Index on Expression – V9

- Queries that include expressions cannot be index only

```
SELECT NAME
        FROM EMP
        WHERE SALARY + COMM < 20000
```

- Can now have indexes created on expressions

```
CREATE INDEX TOTALSAL ON EMP
( SALARY + COMM)
```

- New type of index to help have more efficient use of column expressions
- Some restrictions
  - Cannot be clustered
  - Cannot be primary or foreign key

Index on Expression – V9

- Index on expression allows us to expand the indexability and control of our tables within the database
  - Create indexes on derived columns and formulas
  - Enforce uniqueness on derived columns

I store a date in my table, but need to search on a collection of years

```
CREATE INDEX NAME_IDX1 ON NAME_TBL
(YEAR(BRTH_DTE), LAST_NME);
```

```
SELECT FRST_NME, LAST_NME, BRTH_DTE
FROM   NAME_TBL
WHERE YEAR(BRTH_DTE) IN ('2000', '2002', '2006')
AND    LAST_NME = 'RADY';
```

This is a two column match against the NAME_IDX1 index. Prior to V9 we needed a redundant table column to do this

Index on Expression – V9

- **Allows for a search on a derived value without having to store that value in a table**

List all the accounts that will have a balance under $10,000 in 12 months

```
CREATE INDEX ACCT_IDX_3 ON ACCOUNT
(ACCT_BAL – (PAY_AMT * 12) , ACCT_ID);
```

```
SELECT  ACCT_ID
FROM    ACCOUNT
WHERE ACCT_BAL – (PAY_AMT * 12) < 10000;
```

Prior to V9 this query would scan the entire ACCOUNT table. With version 9 and index can be built to directly answer this question with an index only scan

Conclusions
Challenges and V8/V9 Solutions

Conclusion

- Discussed some very useful V8 features
  - Database (DPSIs, Volatile Tables)
  - Application (Multi-Row Fetch)
  - SQL (Scalar Fullselect, Common Table Expressions)
- Discussed some current challenges with V8
  - Features that work well
  - Features that did not work so well
- Discuss some needs that still exist
  - More optimization features
  - Database availability
- Looked at V9 features
  - Database (UTS, Clone Tables)
  - Application (Native SQL Procedures)
  - SQL (Update/Delete within SELECT
- Discussed usage of new database, application and SQL features of V9
  - Looked at how some of these new features will solve old problems
  - Looked at new performance opportunities

Conclusion (cont..)

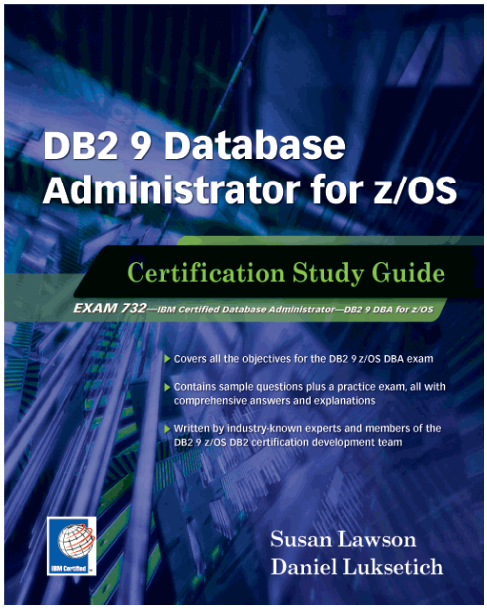
- V8 SQL/Application Performance Features
  - Multi-Row Fetch
  - Recursion
  - Common Table Expressions
  - INSERT within a SELECT
  - Scalar Fullselect
  - Distribution Statistics
  - XML Scalar Functions
  - Multiple Distinct
  - Matching Predicates
  - Parallelism and Sorting
- V8 Database Performance Features
  - Volatile Tables
  - Data Partitioned Secondary Indexes
  - Materialized Query Tables

Conclusion (cont.)

- V9 SQL/Application Features
  - UPDATE/DELETE with a SELECT
  - Optimistic Locking
  - Index on Expression
  - Histogram Statistics
  - Order By and Fetch First in Subselect
  - Instead of Triggers
  - INTERSECT and EXCEPT
  - Skip Locked Data
  - Caseless Expressions
  - Native SQL Procedures
  - XML Support
  - New Functions
  - Distinct Sort Avoidance
- V9 Database Performance Features
  - Better Index Page Splitting
  - Index Compression/Large Pages
  - DPSI Enhancements
  - Clone Tables
  - Append on Insert
  - Truncate Table
  - No Log Tables

DB2 9 for z/OS DBA Certification Guide

- "DB2 9 for z/OS DBA Certification Guide"
- McPress
- January 2008
- Authored By Susan Lawson and Dan Luksetich



DB2 Version 8 for z/OS DBA Certification Guide

- "DB2 for z/OS Version 8 DBA Certification Guide"
- Prentice-Hall
- October 2004
- Authored By Susan Lawson

DB2 V9 Manuals

- DB2 V9.1 for z/OS Administration Guide, SC18-9840-00
- DB2 V9.1 for z/OS Application Programming and SQL Guide, SC18-9841-00
- DB2 V9.1 for z/OS Application Programming Guide and Reference for JAVA SC18-9842-00
- DB2 V9.1 for z/OS Codes, GC18-9843-00
- DB2 V9.1 for z/OS Command Reference, SC18-9844-00
- DB2 V9.1 for z/OS Data Sharing: Planning and Administration, SC18-9845-00
- DB2 V9.1 for z/OS Diagnosis Guide and Reference, LY37-3218-00
- DB2 V9.1 for z/OS Diagnostic Quick Reference, LY37-3219-00
- DB2 V9.1 for z/OS Installation Guide, GC18-9846-00
- DB2 V9.1 for z/OS Introduction to DB2, SC18-9847-00
- DB2 V9.1 for z/OS Licensed Program Specifications, GC18-9848-00
- DB2 V9.1 for z/OS Messages, GC18-9849-00
- DB2 V9.1 for z/OS ODBC Guide and Reference, SC18-9850-00
- DB2 V9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851-00
- DB2 V9.1 for z/OS RACF Access Control Module Guide, SC18-9852-00

DB2 Information on the Web

- IBM
  - Ibm.com
- IBM Software
  - Ibm.com/software
- DB2 Family
  - Ibm.com/software/db2
- DB2 Solutions Directory Applications
  - Ibm.com/developerworks/db2
- "Red Books"
  - ibm. com/ redbooks
- DB2 for z/OS
  - Ibm. Com/software/db2zos
- DB2 Support
  - Ibm.com/software/db2zos/support.html
- DB2 for z/OS Papers
  - ftp://ftp.software.ibm.com/software/data/db2zos
- DB2 Magazine
  - http:// www. db2mag. Com
- DB2 Certification
  - http://www.ibm.com/certify
- DB2 Experts
  - www.db2expert.com