DB2 12 for z/OS Optimizer Sneak Peek

Terry Purcell
IBM Silicon Valley Lab

Session Code: Z2
Wednesday Sept 14th 11:45am – 12:45pm| Platform: DB2 for z/OS

Disclaimer

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Agenda

- DB2 12 Performance Focus
- UNION ALL & Outer Join Enhancements
- Runtime Optimizations
- Sort, Workfile & Sparse index Improvements
- Predicate Optimizations
- Optimizer Cost Model Enhancements
- Plan Stability
- RUNSTATS & Optimizer driven profile updates

DB2 12 Goals

Application Enablement	• IDAA improvements to expand to new use cases • SQL/SQLPL improvements for next wave of applications	• Address key customer requirements to expand use of existing features • Mobile, hybrid cloud, and DevOps enablement
DBA Function	• Relieve table scalability limits • Simplify large table management	• Remove biggest 24x7 inhibitors • Security and compliance improvements
OLTP Performance	• 5-10% CPU reduction with use of in-memory features • 2x increase in Insert throughput for non-clustered	• Remove system scaling bottlenecks for high n-way systems • Serviceability, availability
Query Performance	• 20-30% CPU reduction for query workloads • Improve efficiency by reducing other resource consumption	• 80% UNION ALL performance improvement • Simplify access path management

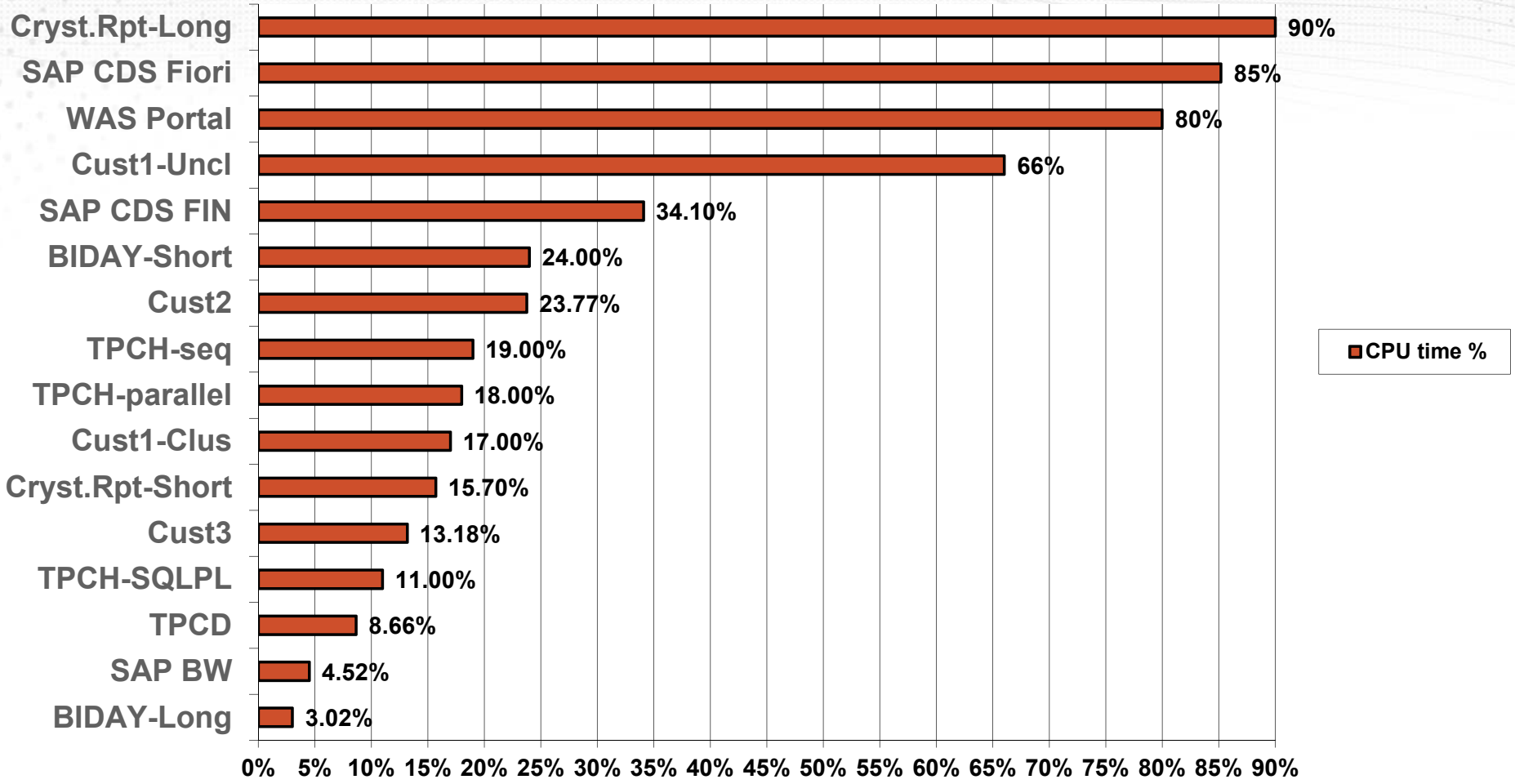
DB2 12 High-level Query Performance Focus

- Query focus based upon newer workloads (modern applications)
  - Complex views or table UDFs
    - UNION ALL
    - Outer joins
    - Join predicates with (stage 2) expressions
  - CASE expressions, CAST functions, scalar functions

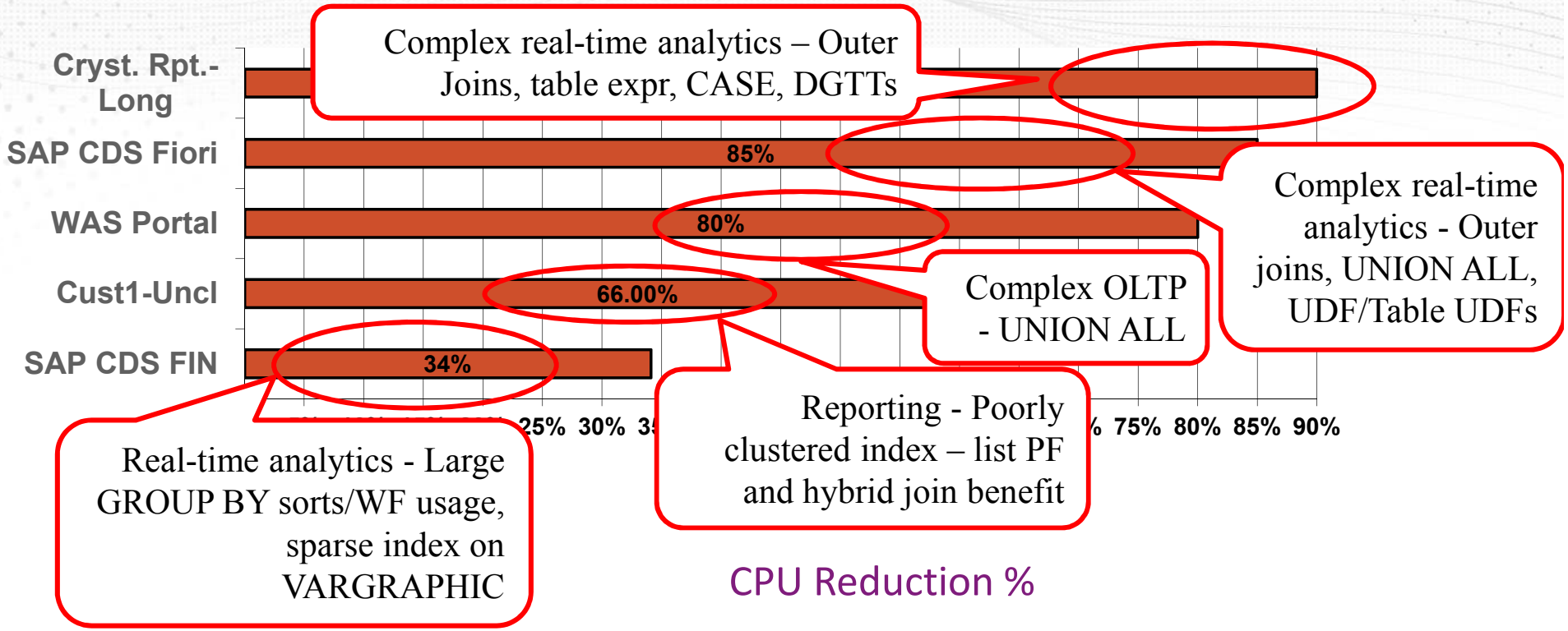
- General Bottlenecks
  - Sort/work file reductions
  - Reducing prepare cost and frequency
  - List Prefetch
  - I/O performance
    - Reduce unnecessary prefetch scheduling

- NOTE: 100% zIIP offload for parallel child tasks

Significant CPU Reduction in DB2 Query Workloads

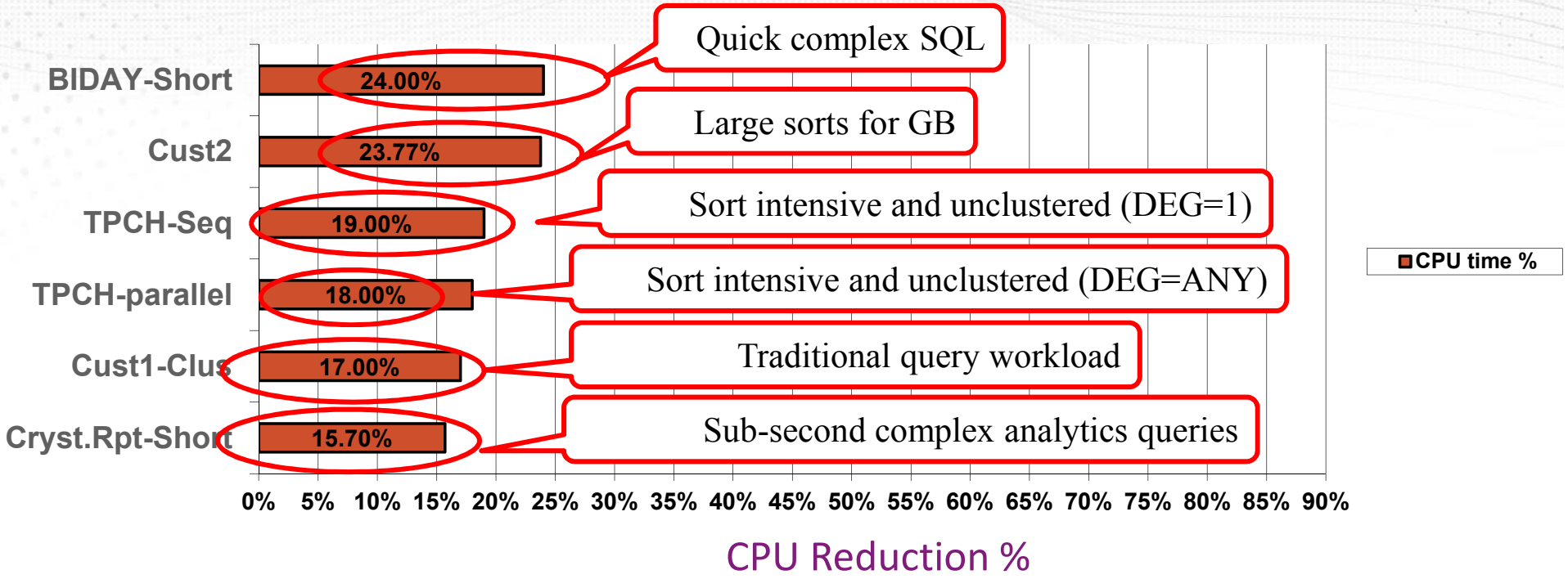


What are the workload characteristics?



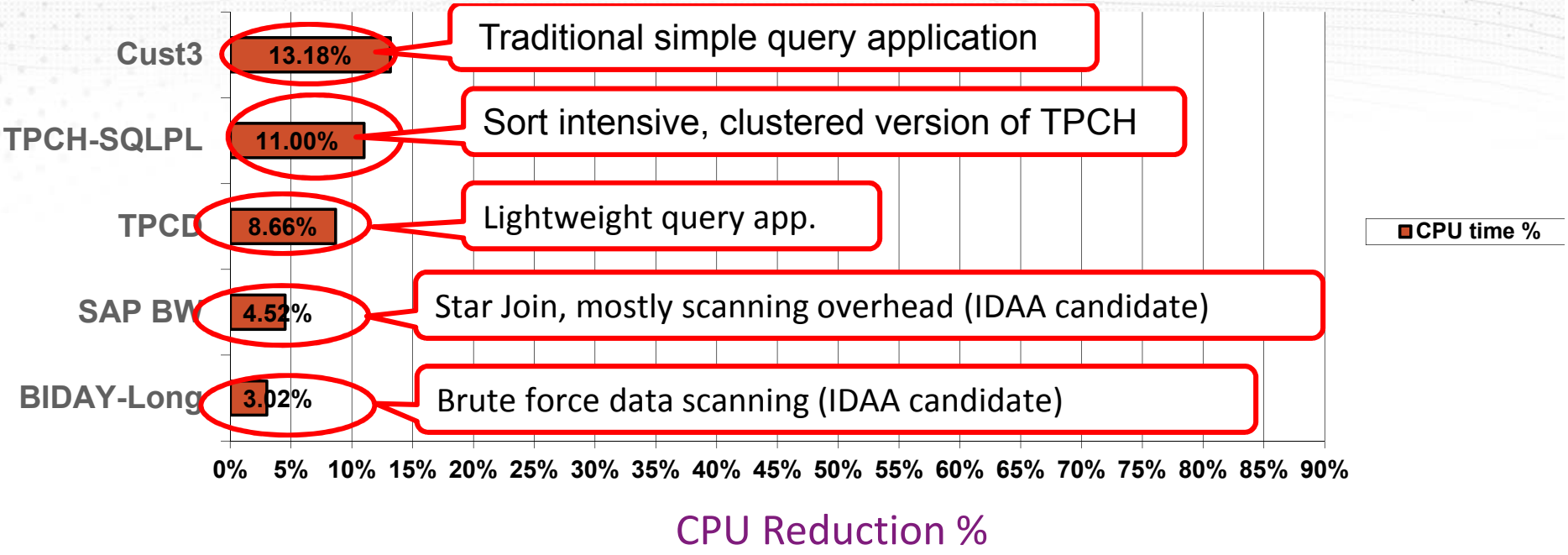
- **30% - 90% reduction in ET and CPU**
  - Complex OJ's, UNION ALL, UDFs & Table UDFs
  - Combinations of Table Expressions, Views and OJ's
  - VARGRAPHIC data type
  - Disorganized data, poor CR indexes
  - ***Nearly 100% NEW Access Paths vs. DB2 11***

What are the workload characteristics?



- **15% - 25% reduction in ET and CPU**
  - Traditional query applications
    - Sort intensive and workfile processing
    - Lots of aggregating
    - ***Many new Access Paths vs. DB2 11***

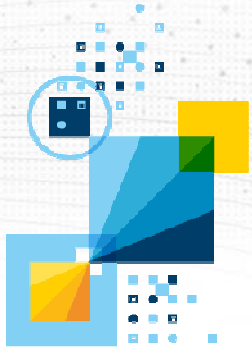
What are the workload characteristics?



- **0%-15% reduction in ET and CPU**
  - Simple query workloads
  - SQL where vast majority of CPU and ET are due to data scanning (IDAA candidates)
  - As % drops - ***Fewer new Access Paths vs. DB2 11***

Give DB2 12 Optimizer a Chance

- **Rebind *without* APREUSE to see full potential**
  - 15+ in house workloads measured
    - Queries with new AP's consistently were the biggest winners
  - Typical performance gains for ***new AP's***
    - 10%-99% reduction in CPU and ET
  - Typical performance gains with ***NO AP change***
    - 0%-20% reduction in CPU and ET
- NOTE: APREUSE provides safety in minimizing risk of regression



UNION ALL and Outer Join Enhancements

UNION ALL and Outer Join

- NOTE: Performance issues are similar with both query patterns
  - Materialization (workfile) usage and inability to apply filtering early
- Union ALL Challenge
  - Replacing a single table with UNION (ALL)
    - Expectation is similar performance
- Outer Join Challenge
  - Replacing an INNER with OUTER join
    - Expectation is similar performance

UNION ALL and Outer Join

- DB2 12 high level solutions
  - Reorder outer join tables to avoid materializations
  - Push join predicates inside UA legs or Outer Join query blocks
  - Sort of outer to ensure sequential access to inner
  - Bypass workfile usage when materialization required
  - Push ORDER BY and FETCH FIRST into UA legs
  - Trim unnecessary columns and tables

Example: Joining tables with Archive Transparency (or any UA views)

Original Query:

```
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;
```

DB2 V11 rewrite:

```
SELECT TX1.C1, TX2.C2 FROM (
        SELECT * FROM T1
        UNION ALL
        SELECT * FROM H1) AS TX1
    LEFT JOIN (
        SELECT * FROM T2
        UNION ALL
        SELECT * FROM H2) AS TX2
    ON TX1.C1 = TX2.C1;
```

Materialization (all cols, all rows)

Materialization (all cols, all rows)

Join of workfiles

Example: V12 Joining tables with Archive Transparency cont.

Original Query:

```
SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;
```

Cost based

DB2 V12 rewrite:

```
SELECT TX1.C1, TX2.C2 FROM (
        SELECT T1.C1 FROM T1 UNION ALL
        SELECT H1.C1 FROM H1) AS TX1
  LEFT JOIN TABLE (
        SELECT T2.C1 FROM T2
        WHERE T2.C1 = TX1.C1
        UNION ALL
        SELECT H2.C1 FROM
        WHERE H2.C1 = TX1.C1) AS TX2
  ON 1=1;
```

Pipelined

Join pushed down to base tables And Inner is pipelined

No workfiles to be joined

Example: V12 Sort Outer

Original Query:

SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;

Cost based

Pipelined

DB2 V12 rewrite:

```
SELECT TX1.C1, TX2.C2 FROM (
    SELECT T1.C1 FROM T1 UNION ALL
    SELECT H1.C1 FROM H1) AS TX1
LEFT JOIN TABLE(
    SELECT T2.C1 FROM T2
    WHERE T2.C1 = TX1.C1
    UNION ALL
    SELECT H2.C1 FROM
    WHERE H2.C1 = TX1.C1) AS TX2
ON 1=1;
```

Can be sorted to match inner indexes

Join pushed down to base tables

Index on T2.C1

Index on H2.C1

No workfiles to be joined

Example: V12 Sparse index

Original Query:

SELECT T1.C1, T2.C2 FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1;

Cost based

Pipelined

DB2 V12 rewrite:

```
SELECT TX1.C1, TX2.C2 FROM (
        SELECT T1.C1 FROM T1 UNION ALL
        SELECT H1.C1 FROM H1) AS TX1
    LEFT JOIN TABLE(
        SELECT T2.C1 FROM T2
        WHERE T2.C1 = TX1.C1
        UNION ALL
        SELECT H2.C1 FROM
        WHERE H2.C1 = TX1.C1) AS TX2
    ON 1=1;
```

Join pushed down to base tables

Build sparse Index on T2.C1

Build sparse Index on H2.C1

No workfiles to be joined

Example: V11 ORDER BY / FFNR with UNION ALL

View:

```
CREATE VIEW V1 AS (  SELECT * FROM T1   UNION ALL
                     SELECT * FROM T2   UNION ALL
                     SELECT * FROM T3);
```

Original query:

```
SELECT C1 FROM V1 ORDER BY C1 FETCH FIRST 10 ROWS ONLY;
```

DB2 V11 rewrite:

```
SELECT C1 FROM (
    SELECT * FROM T1  UNION ALL
    SELECT * FROM T2  UNION ALL
    SELECT * FROM T3 ) AS V1
ORDER BY C1
FETCH FIRST 10 ROWS ONLY;
```

Materialization (all rows)

Sort (all rows)

10 rows fetched from workfile

Example: V12 ORDER BY / FFNR with UNION ALL

View:

```
CREATE VIEW V1 AS (  SELECT * FROM T1   UNION ALL
                     SELECT * FROM T2   UNION ALL
                     SELECT * FROM T3);
```

Original query:

```
SELECT C1 FROM V1 ORDER BY C1 FETCH FIRST 10 ROWS ONLY;
```

DB2 V12 rewrite:

ORDER BY and FFNR Pushed down to base tables

```
SELECT C1 FROM (
    SELECT C1 FROM T1 ORDER BY C1 FETCH FIRST 10 ROWS ONLY
    UNION ALL
    SELECT C1 FROM T2 ORDER BY C1 FETCH FIRST 10 ROWS ONLY
    UNION ALL
    SELECT C1 FROM T3 ORDER BY C1 FETCH FIRST 10 ROWS ONLY) AS V1
ORDER BY C1
FETCH FIRST 10 ROWS ONLY;
```

Merge of 30 rows Fetch first 10

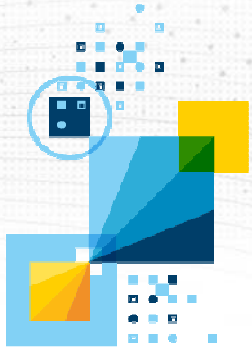
Extended LEFT JOIN pruning

- DB2 10 delivered LEFT OUTER JOIN table pruning
  - If right table guaranteed unique (due to unique index or DISTINCT/GROUP BY) and no columns required for final result

```
SELECT T1.*
FROM T1 LEFT JOIN T2
ON T1.C1 = T2.UNIQUE_COL;
```

```
SELECT DISTINCT T1.*
FROM T1 LEFT JOIN T2
ON T1.C1 = T2.C1;
```

- DB2 12
  - Extends table pruning to
    - Views, table expressions
      - Where no columns required and no duplicates returned
  - Improves pruning of unreferenced columns



Runtime Optimizations

Generic Query Challenge

- Challenge for query optimizers
  - Filtering could change each execution
    - Making it impossible to choose the 1 best access path

```
SELECT *
FROM CUSTOMER
WHERE LASTNAME LIKE ?
    AND FIRSTNAME LIKE ?
    AND ADDRESS LIKE ?
    AND CITY LIKE ?
    AND ZIPCODE BETWEEN ? AND ?
```

    - Fields NOT searched by user will populate with the whole range:
      - LIKE '%' or BETWEEN 00000 AND 99999
    - Fields searched will use their actual values
      - LIKE 'SMITH' or BETWEEN 95141 AND 95141

Runtime adaptive index

- Allow list prefetch based plans to quickly determine index filtering @ execution time
  - Adjust (adapt) @ execution time based on determined filtering
    - Without requiring REOPT(ALWAYS)
    - For list prefetch or multi-index OR-ing
      - Earlier opportunity to fallback to tablespace scan if large % of table to be read
    - For multi-index AND-ing
      - Reorder index legs from most to least filtering
    - Early-out for non-filtering legs, and fallback to rscan if no filtering
  - Quick evaluation done based upon literals used (e.g. LIKE '%')
  - Further (more costly) evaluation of filtering deferred until after 1 RID block retrieved from all participating indexes
    - Provides better optimization opportunity while minimizing overhead for short running queries

Runtime Adaptive Index Solution

- Not limited to the search screen challenge
  - Any query where there exists high uncertainty in the optimizer's estimates
    - Range predicates
    - JSON, Spatial & Index on expression

```
SELECT * FROM TAB1 WHERE COL1 < ? AND COL2 < ? AND COL3 < ?
INDEXES: IX1(col1), IX2(col2), IX3(col3)
```

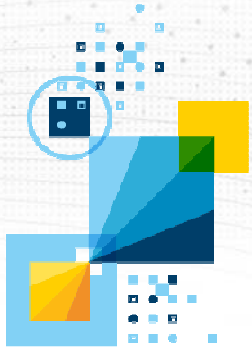
  - Performance
    - List PF up to 23% CPU reduction (when failover to rscan needed)
    - Multi-index OR up to 20% CPU reduction (when failover to rscan needed)
    - Multi-index AND up to 99% CPU reduction for re-ordering to put most filtering leg 1st

Solution to prior RID-failover to WF

- DB2 10 added RID failover to WF
  - However, it did not address the filtering estimation issue
  - Some customers needed to alter MAXTEMPS_RID=NONE
    - To disable failover to WF
- DB2 12
  - Recommendation to keep default MAXTEMPS_RID=NOLIMIT
    - Due to “runtime adaptive index” improvements

UDF Caching

- Improve performance of UDFs by caching results by
  - maintaining a hash table that stores UDF result
    - hash key is the UDF input values
  - Scope
    - UDFs that are DETERMINISTIC and NO EXTERNAL ACTION
    - Avoid invoking UDF and input expression again for the same input
    - Caching is within the scope of an individual UDF instance only (cache is not shared)
  - Features
    - all datatypes (except XML,LOB,ARRAY) and expressions with up to 1 column
    - adaptively turn off caching if no benefit seen (check every 1024 executions)
  - Performance
    - Could reduce UDF time up to 99% for low cardinality inputs



Sort, Workfile & Sparse Index Improvements

Sort minimization for partial order with FETCH FIRST

- Problem
  - With ORDER BY & FETCH FIRST n ROWS ONLY
    - Only avoid ORDER BY sort if an index is chosen that supports the ORDER BY
      - Allowing on the FFnR number of rows to be fetched
- DB2 12
  - Can utilize an index that provides order on the leading columns of the ORDER BY
    - Reducing the number of rows fetched/processed if sort cannot be avoided

Example - Partial order with FETCH FIRST

- Example

```
SELECT * FROM T1
ORDER BY C1, C2
FETCH FIRST 10 ROWS ONLY

INDEX1 (C1)
Index entries: 1,1,1,2,2,2,3,3,3,4,4,4,5……999999
```

- Pre DB2 12
  - Fetch all rows and sort into C1, C2 sequence
    - millions of rows in this example
- DB2 12
  - Once 10th row is reached, then fetch until change of C1
    - 12 rows fetched/sorted in this example

Sort avoidance/minimization

- Sort avoidance for OLAP functions with PARTITION BY clause
  - ROW_NUMBER, RANK, DENSE_RANK
- Remove constants from the GROUP BY/DISTINCT sort key
- Reduce sort key redundancy
- Bypass sort if data detected in order
- In-memory sort
  - Continuing theme from V9 – 11
    - V11 avoids final write to WF
  - Extends “avoid write to WF” for intermediate sorts
    - Exceptions – Hybrid Join, Sort-merge Join sort new, parallelism

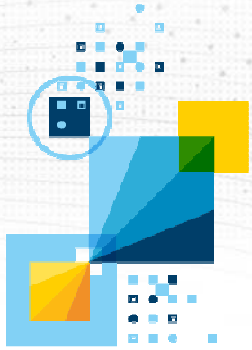
Improve GROUP BY/DISTINCT sort performance

- Pre DB2 12
  - Maximum number of nodes of the sort tree is 32,000
    - Less for longer sort keys (limited by zparm SRTPOOL)
  - DB2 9 added hashing as input to GROUP BY/DISTINCT sort

- DB2 12
  - Sort tree (and hash entries) can grow to
    - 512,000 nodes (non-parallelism sort) or 128,000 (parallel child task sort)
      - Default 10MB SRTPOOL can support 100,000 nodes for 100 byte row
        - Increasing SRTPOOL in DB2 12 can have greater effect than pre DB2 12
  - Performance
    - Up to 70% CPU saving when sort can be contained in memory

Sparse index improvements

- Sparse index support for VARGRAPHIC datatype
- Improved memory allocations when multiple sparse indexes in query
- Memory reductions
  - Avoid duplicating key information when key=data and fixed length key
  - Trim trailing blanks for VARCHAR/VARGRAPHIC
  - Trim prefix if all keys have the same prefix



Predicate Optimizations

Sort on expression for join

- Expressions (scalar functions, arithmetic etc) as join predicates are often stage 2
  - Exception is expression on the outer (without sort) for nested loop join (NLJ)

- DB2 12
  - Allow resolution of expr before sort to support join support for
    - Expression on inner and sparse index (without sort on outer)

```
SELECT …
FROM T1, T2
WHERE       T1.col1 = T2.col1
AND         T1.col2 = SUBSTR(T2.col2,1,10) …
```

  - Does not apply to
    - CASE expressions, DECFLOAT, different CCSIDs, FULL OUTER JOIN

Merge for Table UDFs similar to Views

- Merge rather than materialization for Table UDFs
  - Defined as DETERMINISTIC and NO EXTERNAL ACTION
  - Similar to performance of views
- Support indexability of predicates passed in as TUDF parameters

```
CREATE FUNCTION tpchudf(in_date1 CHAR(10), in_date2 CHAR(10))
…
SELECT O_ORDERPRIORITY, COUNT(*)
FROM ORDERS
WHERE  O_ORDERDATE >= DATE(in_date1) <<< Was STAGE 2. Now indexable
   AND O_ORDERDATE <  DATE(in_date2) <<< Was STAGE 2. Now indexable


SELECT * FROM table(tpchudf('1993-07-01', '1993-10-01')) as TAB1;
```

VARBIN datatype & scalar function indexability

- Pre-DB2 12
  - Stage 2 for VARBIN/BINARY predicates with mismatched lengths
- DB2 12
  - CAST VARBIN/BINARY predicates when operand lengths don't match
    - Predicates now indexable
  - Support indexability for IOE built on VARBIN/BINARY datatypes
    - Example
      - IOE for COLLATION_KEY with collation name 'UCA410_LDE' (German)

```
CREATE INDEX EMPLOYEE_NAME_SORT_KEY ON EMPLOYEE
(COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600));

SELECT  *
FROM   EMPLOYEE
WHERE  COLLATION_KEY(LASTNAME, 'UCA410_LDE', 600) = < ?
```

Row permission correlated subquery indexability

- Pre DB2 12
  - Correlation predicates in child correlated subquery are stage 2 on row permissions for Insert and Update.
- DB2 12
  - Support indexability for correlated subqueries on row permissions for Insert/Update

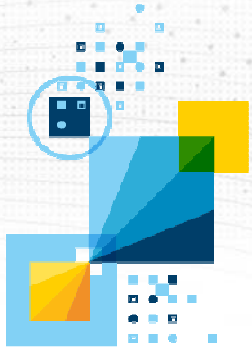
```
CREATE PERMISSION RP1 ON LINEITEM T1 FOR ROWS
WHERE EXISTS
(SELECT 1 FROM ORDER T2
 WHERE T1.ORDERKEY=T2.ORDERKEY) ENFORCED FOR ALL ACCESS
ENABLE

- Insert row into table LINEITEM

- UPDATE LINEITEM SET COMMENT='?' WHERE ORDERKEY = ?;
```

IN-list Predicate optimizations

- DB2 12 IN-list predicate enhancements
  - Index matching
    - Improvement to matching IN-list performance for poor clustering index
      - Can choose to accumulate 32 RIDs per list PF request
        - Rather than 1 list PF request per IN-list entry
        - Similar to hybrid join (SORTN=N)
  - Index screening predicates
    - Optimized IN-list performance with > 128 elements
      - Retrofitted to DB2 10 & 11 via APAR PI41598
    - Removes limitation that IN-lists cannot be index screening for range-list access



Optimizer Cost Model Enhancements

Extend NPGTHRSH to default stats

- NPGTHRSH zparm prioritizes matching index access over tablespace scan
  - If NPAGESF is < NPGTHRSH
  - NPGTHRSH recommendation is generally 10 or less
- Pre DB2 12
  - Default stats (-1) = 501 pages. Thus default stats may result in tablespace scan chosen.
- DB2 12
  - Compare -1 to NPGTHRSH rather than 501
  - Applies when
    - table and indexes both have default stats
    - table has stats but one or more indexes have default stats

List Prefetch and Hybrid Join

- Optimizer enhancements to list prefetch due to improved runtime
  - Adaptive index has better runtime estimation of actual predicate filtering
    - Allowing earlier fallback to tablespace scan when poor filtering
  - RID sort performance improvements
- Improved Hybrid Join (HBJ) as choice within parallelism
  - Enables HBJ (without sort new) for all clusterratios based upon cost
- Significant performance improvement possible for unclustered data

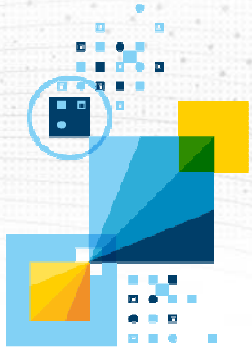
Bind/Prepare enhancements

- Index probing
  - Used for dynamic SQL with literals (or REOPT) when either...
    - Table is VOLATILE or qualifies for NPGTHRSH
    - Predicate or index is estimated to qualify no rows
  - DB2 12 Improves accuracy and performance of index probing
- Access path selection with larger number of indexes
  - DB2 12 discards poor index candidates early
    - Improving prepare performance.
    - Potentially ensuring indexes with greater matching columns survive

Improve Filter Factor for CURRENT DATE/TIMESTAMP

- Problem
  - Predicates including special registers CURRENT DATE/TIMESTAMP are treated similar to host variables/parameter markers
    - Result is a default Filter Factor estimate
- DB2 12
  - COL op CURRENT DATE / TIMESTAMP (incl non-col expression)
    - Will resolve the expression to actual value for filter factor purposes.

```
SELECT * FROM T1
WHERE SALES_DT > CURRENT_DATE - 7 DAYS
  AND SALES_DT < CURRENT_DATE;
```



Plan Stability

Dynamic Plan Stability

- Provide “static SQL” like performance stability for repeating cached dynamic SQL
  - Store dynamic SQL and cache structures in the DB2 catalog
  - Load cache from catalog on cache miss, catalog hit

- Allow more stable, predictable query performance across...
  - Exit / re-entry to statement cache
  - System recycle
  - RUNSTATS
  - DB2 maintenance
  - Release migration
  - Across members of data sharing group

  - Heavy “cache-load” periods can see significant performance improvement – full prepares replaced with loading cache from disk.

Static Plan Stability Enhancements

- Free inactive package copies while package is allocated and in use
  - Allow FREE of INACTIVE only
- Allow selective FREE of ORIGINAL or PREVIOUS
- APREUSE source
  - Supports APREUSE of PREVIOUS or ORIGINAL in 1 step

Support statement concentration via BIND parameter

- [RE]BIND PACKAGE

```
...
>>-[RE]BIND PACKAGE-------------------------------------------------->
...
>>-+--------------------   ----+------------------------------>
   |                   .-NO--.   |
   '-CONCENTRATESTMT(-+-----+-)-'
                     '-YES-'
```

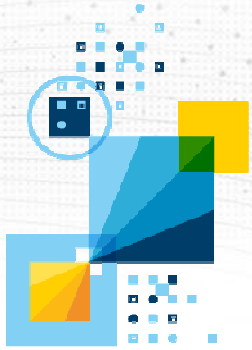
CONCENTRATESTMT:

NO

- Do not enable statement concentration. Literals in dynamic SQL statements are not replaced.This is the default value.

YES

- Enable statement concentration. Literals in dynamic SQL statements are replaced by '&'.

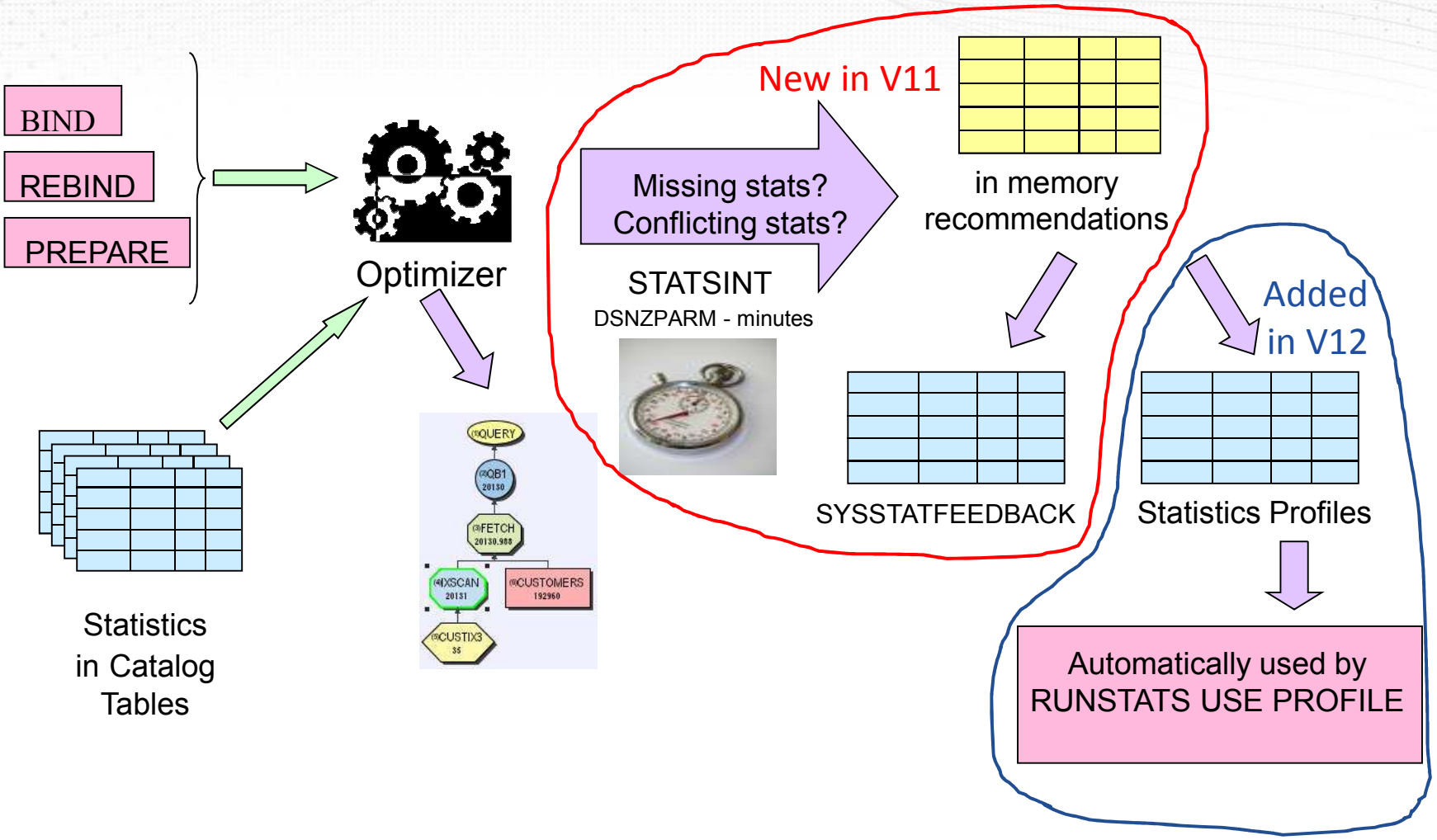


RUNSTATS & Optimizer generated profile updates

Runstats enhancements for SQL performance

- Clusterratio formula improvements
- DB2 12 default is that RUNSTATS will NOT invalidate cache
  - Exception UPDATE NONE REPORT NO
- Profile support for inline stats
- Automated COUNT for FREQVAL
  - Allow DB2 to collect until values no longer skewed (rather than count 10)
- Optimizer to automatically update statistics PROFILE
  - Specify USE PROFILE to automatically pick up optimizer recommendations

DB2 12 – Optimizer externalization of missing statistics



Terry Purcell

IBM Silicon Valley Lab

tpurcel@us.ibm.com

Please fill out your session evaluation before leaving!

Session: Z2

Title: DB2 12 for z/OS Optimizer Sneak Peek