Z30 – Part B

Fundamentals of DB2 Query Optimization

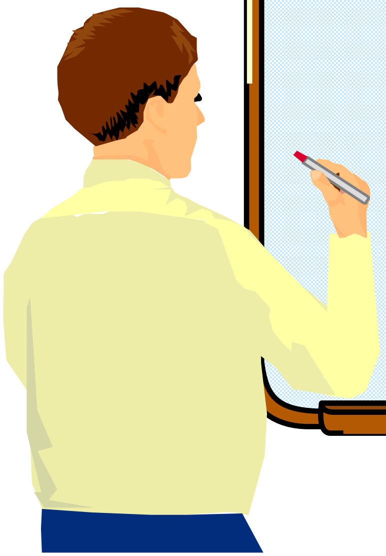
Gene Fuh, IBM Silicon Valley Laboratory

IBM DB2 Information Management **Technical Conference**

Sept. 20-24, 2004

Las Vegas, NV

Agenda



Part A

Session 1: Overview

Session 2: Access path and explain table

Session 3: DB2 Runtime Architecture and predicate application

Session 4: Access methods

Part B

Session 5: Join methods

Session 6: Query transformation

Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

Agenda

Part A

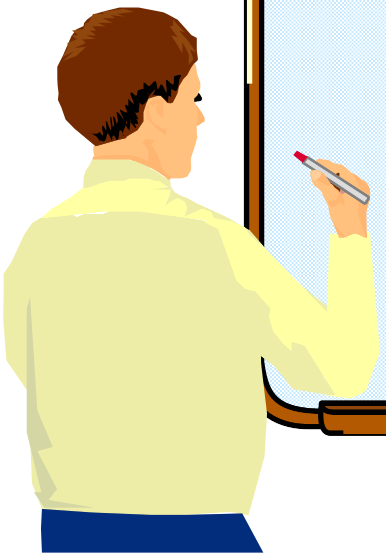
Session 1: Overview

Session 2: Access path and explain table

Session 3: DB2 Runtime Architecture and predicate application

Session 4: Access methods

Part B



Session 5: Join methods

Session 6: Query transformation

Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

Join Method Execution

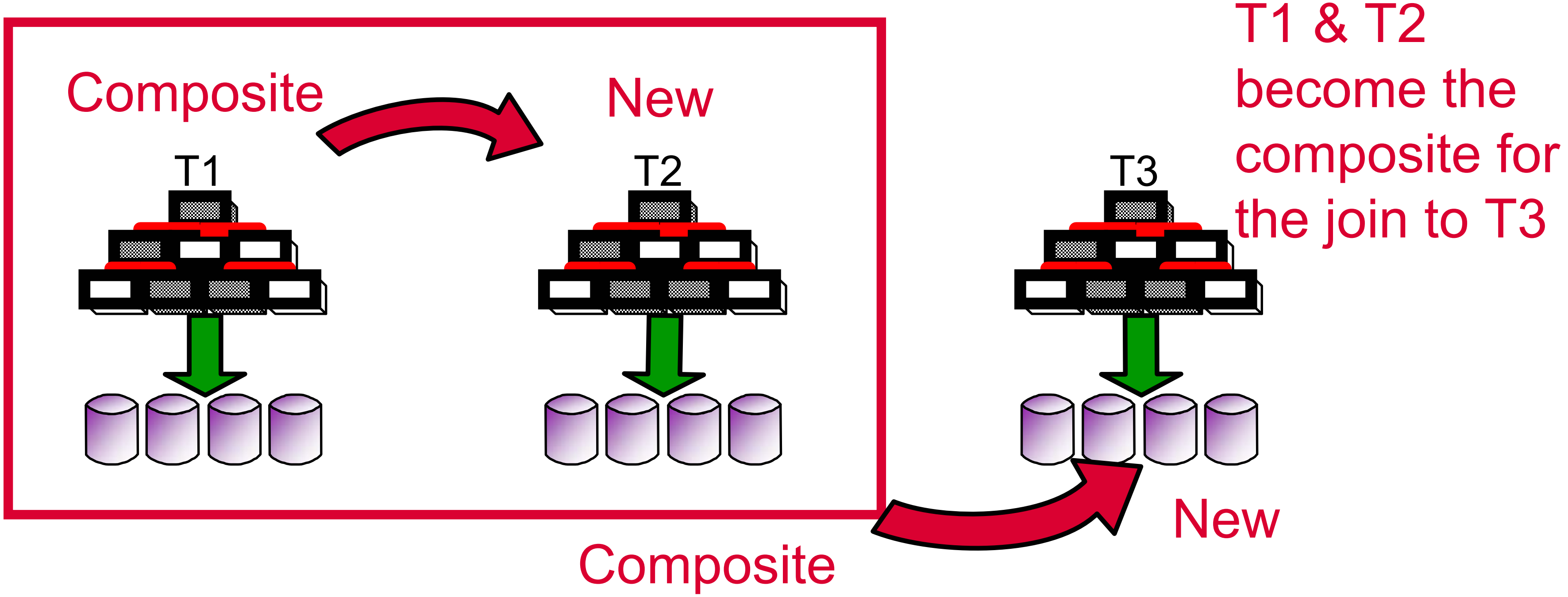
- **This Section:**
  - **Objectives**
    - To cover the three join methods used for processing SQL containing table joins.
  - **Join Methods**
    - Nested Loop Join
    - Sort Merge Join
    - Hybrid Join

    - NOTE: The fourth join method, Star Join, will not be discussed in this presentation

Table Join Terminology

- **Composite Table**
  - **Outer table of the join**
    - **In a two table join, this is the first table accessed**

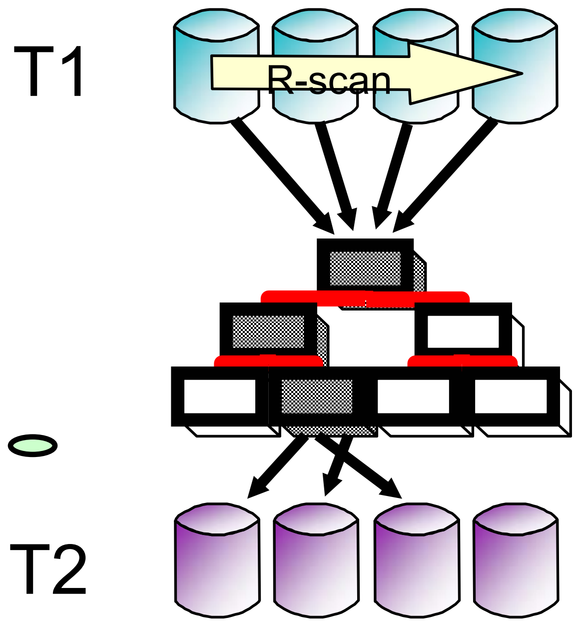
- **New Table**
  - **Inner table of the join**
    - **In a two table join, this is the second table accessed**



Join Methods - Nested Loop Join

▪ Nested Loop Join (NLJ)

- Access outer (composite) table using efficient single table access
- For each qualifying outer table row access the inner table using efficient single table access
- Join the results




```
SELECT   T1.C4, T2.C6
FROM     T1, T2
WHERE    T1.C1 = T2.C1
AND      T1.C4 > 1 ;
```

Nested Loop Join

SELECT *
FROM DSN8710.EMP E JOIN DSN8710.PROJ P
ON E.WORKDEPT = P.DEPTNO
WHERE E.WORKDEPT IN ('A00', 'B01', 'C01')

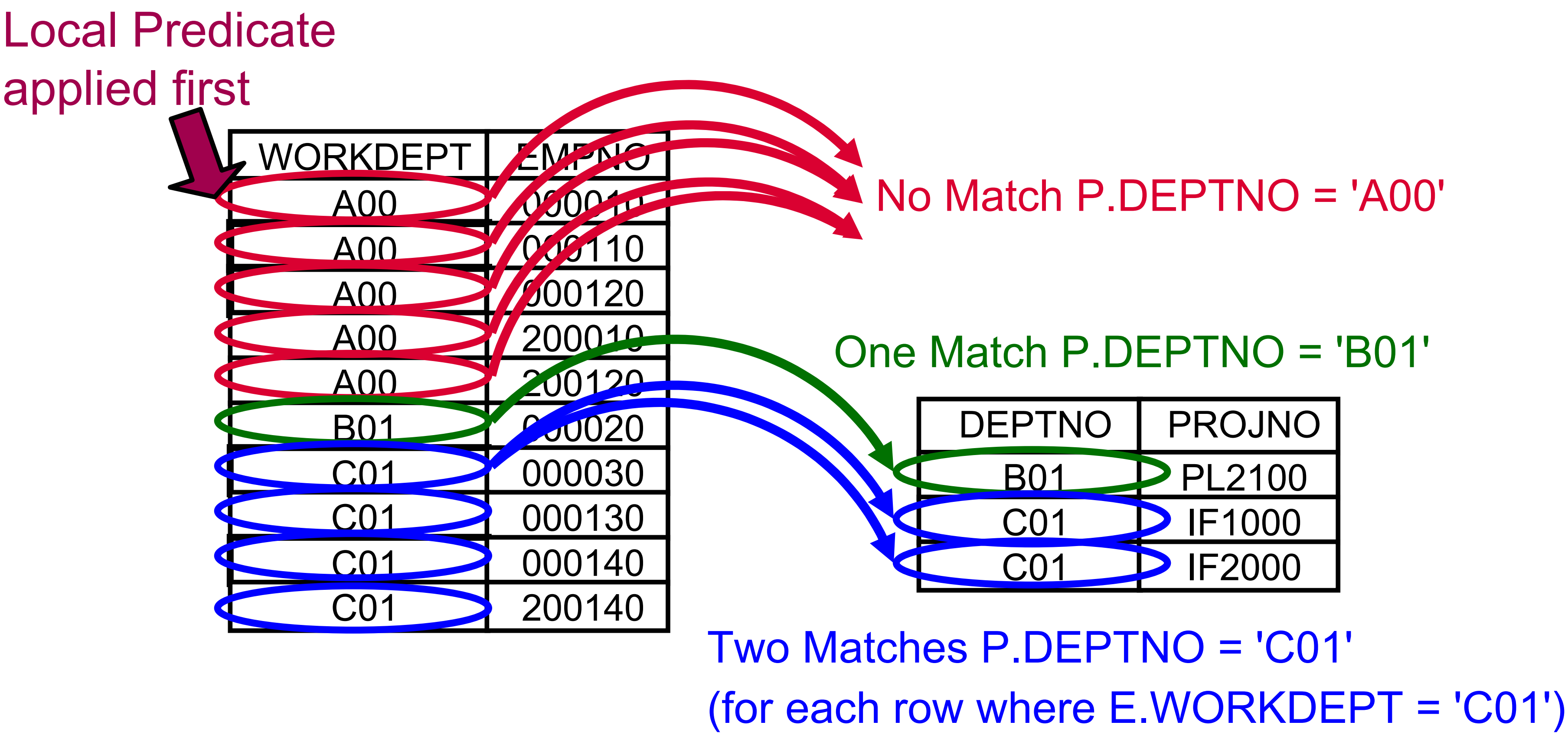
QQP	M T H	TNAME	A T Y	M C O	A_NM	IX O	SORT NCCCC J UJOG	CORR _NM	QB_TYP	P Q B	TB_T YP
01-01-01	0	EMP	N	1	EMPX2	N		E	SELECT	0	T
01-01-02	1	PROJ	I	1	PROJX2	N		P	SELECT	0	T

IN list/Index access, 1 matchcol for local and join predicate

Nested Loop Join

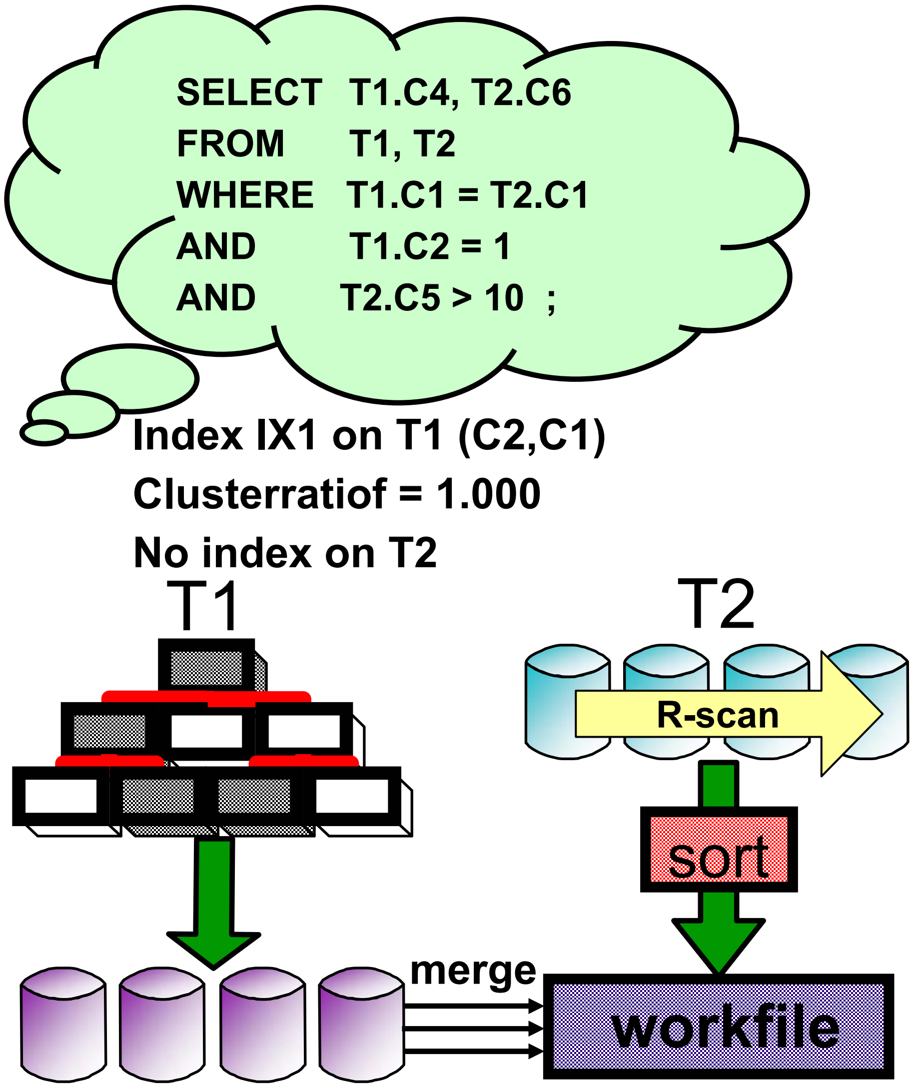
Nested Loop Join

```
SELECT *
FROM      DSN8710.EMP E JOIN DSN8710.PROJ P
ON        E.WORKDEPT = P.DEPTNO
WHERE     E.WORKDEPT IN ('A00', 'B01', 'C01')
```



Join Methods - Sort Merge Join

- **Sort Merge Join (SMJ)**
  - Also known as Merge Scan Join.
  - Access inner/outer table using efficient single table access and apply eligible S1/S2/SubQry predicates
  - Sort inner/outer tables (can avoid sort if index provides ordering)
  - Inner table always written to workfile
  - Merge filtered, sorted inputs



Sort Merge Join

```
SELECT *
FROM    DSN8710.EMP E FULL JOIN DSN8710.DEPT D
ON      E.WORKDEPT = P.DEPTNO
```

Full Join used to force SMJ

QQP	MTH	TNAME	ATYP	MCOI	A_NM	IXO	SORT NCCCC J UJOG	PF	MJC	CORR _NM	JT
01-01-01	0	EMP	I	0	EMPX2	N				E	
01-01-02	2	DEPT	R			N	YNNNN	S	1	D	F

Sort Merge Join

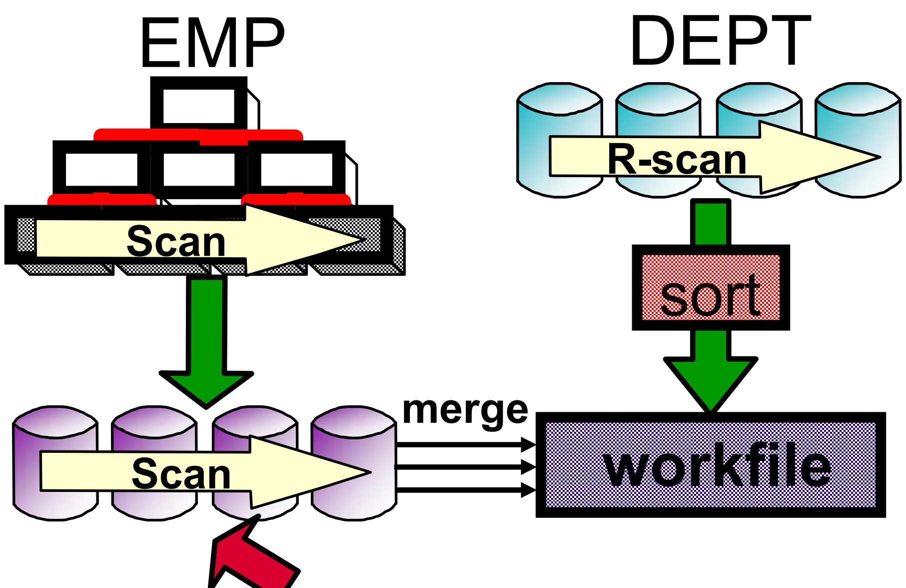
SORTN_JOIN = 'Y'.

Merge Join Cols = 1

Sort Merge Join - Sort New

QQP	MTH	TNAME	ATYP	MCOI	A_NM	IXO	SORT NCCCC J UJOG	PF	MJC	CORR_NM	JT
01-01-01	0	EMP	I	0	EMPX2	N				E	
01-01-02	2	DEPT	R			N	YNNNN	S	1	D	F

SORTN_JOIN = 'Y'

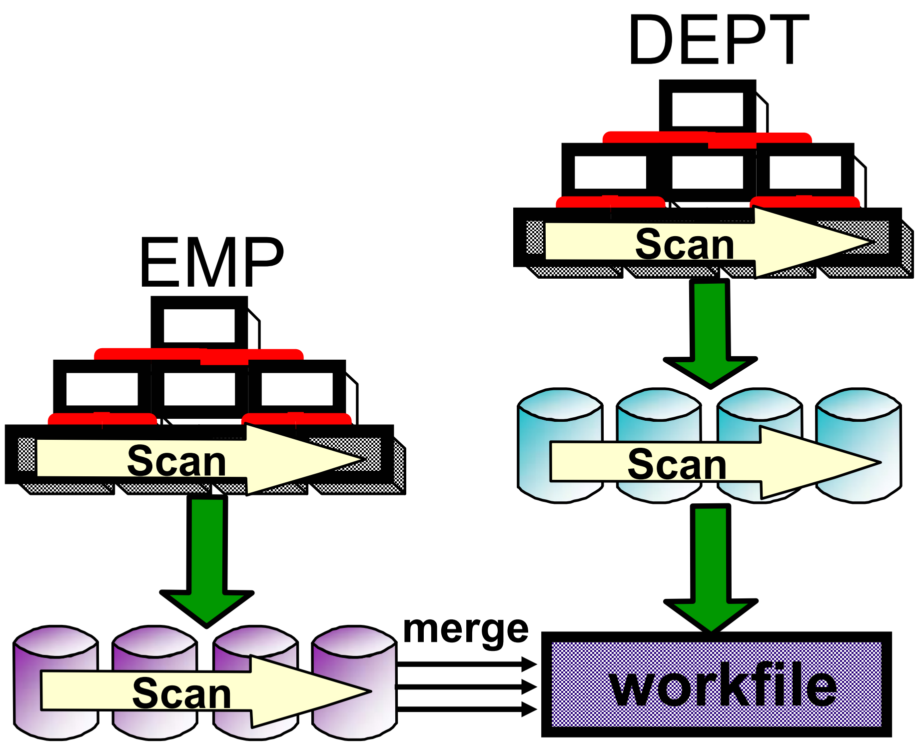


Data accessed in index sequence

- Read DEPT using R-scan into a workfile
- Sort workfile into join col seq
- Access EMP using non-matching index scan (to avoid sort)
- Match/merge EMP with workfile (while reading EMP)

Sort Merge Join - Sort None

QQP	MTH	TNAME	ATYP	MCOL	A_NM	IXO	SORT NCCCC J UJOG	PF	MJC	CORR _NM	JT
01-01-01	0	EMP	I	0	EMPX2	N	NNNNN			E	
01-01-02	2	DEPT	I	0	DEPTX1	N	NNNNN		1	D	F

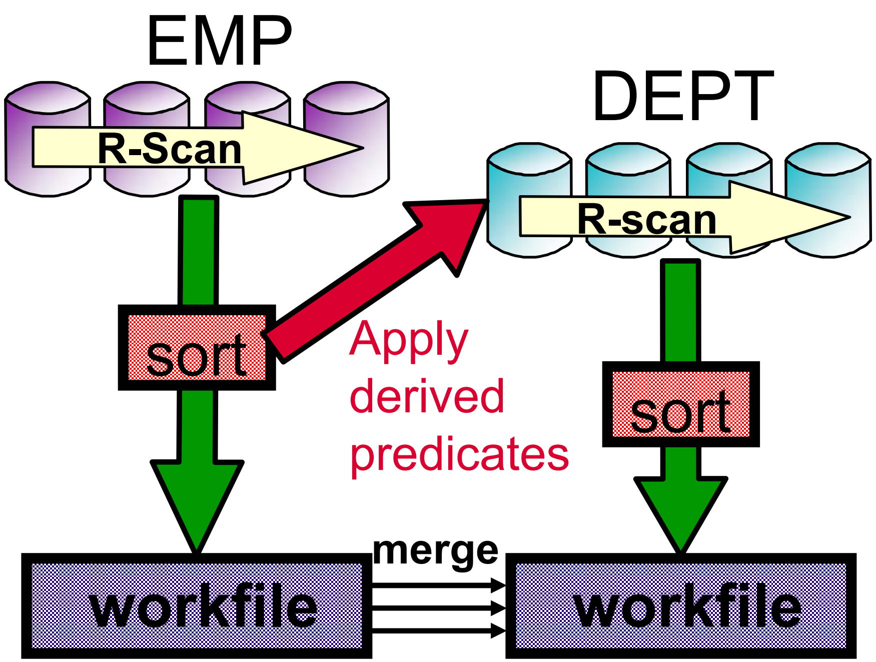


- Read DEPT using non-matching index scan (to avoid sort) into a workfile
- Access EMP using non-matching index scan (to avoid sort)
- Match/merge EMP with workfile (while reading EMP)

Sort Merge Join - Sort Both

QQP	MTH	TNAME	ATYP	MCOL	A_NM	IXO	SORT NCCCC J UJOG	PF	MJC	CORR_NM	JT
01-01-01	0	EMP	R			N		S		E	
01-01-02	2	DEPT	R			N	YNYNN	S	1	D	

SORTN_JOIN & SORTC_JOIN = 'Y'

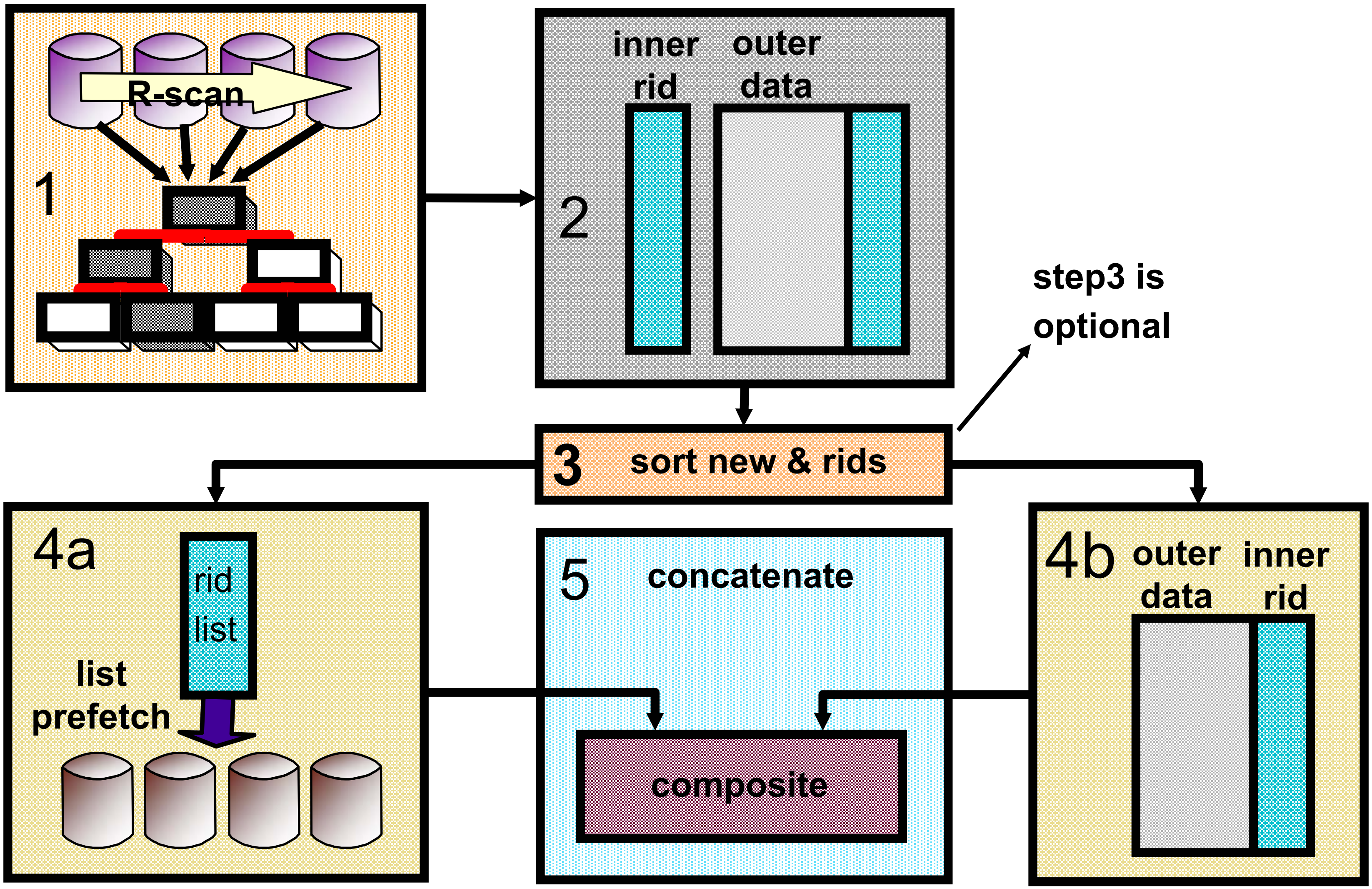


- Read EMP using R-scan into a workfile
- Sort EMP workfile into join col seq
- Derive range predicates from EMP sort (not for FULL JOIN)
- Read DEPT using R-scan into a workfile
- Apply predicates derived from EMP (while reading DEPT)
- Sort DEPT workfile into join col seq
- Match/merge EMP workfile with DEPT workfile

Join Methods - Hybrid Join

- Hybrid Join (HYB)
  - Apply only to an inner join and requires an index on the join column(s) of the inner table
  - Access the outer table using efficient single table access
  - Optionally sort the outer table into inner table join sequence
  - Join the outer table with RIDs from the inner table index --> workfile
  - Optionally sort the workfile into RID sequence (outer table data + inner table RIDs)
  - Retrieve the inner table data with list prefetch
  - Concatenate inner table data with outer table data

Hybrid Join Steps



Hybrid Join - Sorting

In addition to SORTN_JOIN as seen on previous page......

QQP	M T H	TNAME	A T Y P	M C O L	A_NM	IX O	SORT NCCCC J UJOG	
01-01-01	0	EMP	R	0		N		
01-01-02	4	DEPT	I	1	DEPTX1	N	NNNNN	L

No sort due to high clusterratio index. Inner table accessed with List Prefetch, without RID sort.

No sort

QQP	M T H	TNAME	A T Y P	M C O L	A_NM	IX O	SORT NCCCC J UJOG	P F
01-01-01	0	EMP	R	0		N		
01-01-02	4	DEPT	I	1	DEPTX3	N	YNYNN	L

Composite (outer) table sorted before inner table access. Sort on inner required also.

SORTN_JOIN & SORTC_JOIN = 'Y'

Join Methods - Hybrid Join

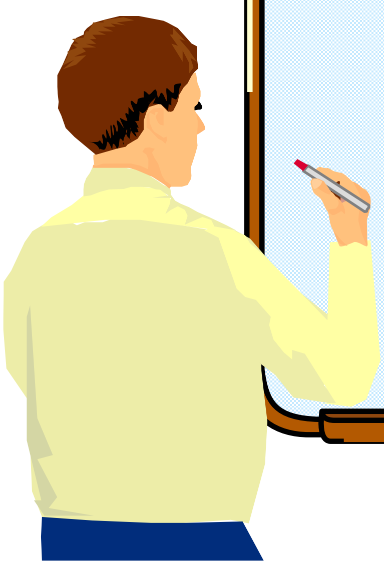
- **Hybrid Join Notes**

  - **Better utilization of List Prefetch than Nested Loop Join**
    - **Inner table is accessed once using List Prefetch, rather than once for each outer row.**

  - **Outer table local predicates applied before the join/sort**
    - **All indexable, stage 1 & 2 (including subqueries) are applied on the outer table before a composite sort (if required) and before the inner table is accessed**

  - **Inner table predicates applied before/after join/sort**
    - **All index matching predicates are applied as the inner table index is accessed, and before the sort if required.**
    - **Non-index matching predicates are applied after data access (thus after sort).**

Agenda



Part A

Session 1: Overview

Session 2: Access path and explain table

Session 3: DB2 Runtime Architecture and predicate application

Session 4: Access methods

Part B

Session 5: Join methods

Session 6: Query transformation

Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

Query Transformation

- **This section introduces:**
  - **Purpose of transformations**
    - **Unlock more possible access path choices**
    - **Allow cost model to estimate and choose most efficient**
  - **Transformations**
    - **Predicate transformations**
    - **Join transformations**
    - **View / table expression transformations**
    - **Distribution and pruning**

Query Transformation

- **Predicate transformations**
  - **In-list / between ==> equal**
  - **OR ==> In-list**
  - **Predicate transitive closure**
  - **Predicate pushdown**

In-list / between to equals

- In-list / between to equals
  - Between / inlist can stop matching in index earlier
    - In-list not candidate for PTC
  - More statistics are usable
  - Examples:
    - WHERE C1 IN (1) ==> WHERE C1 = 1
    - WHERE C1 BETWEEN 1 AND 1 ==> WHERE C1 = 1

Or to in-list

- OR ==> In-list
  - Candidate for single index in-list access
  - WHERE (C1 = 1 OR C1 = 2 OR C1 = 3) ==> C1 IN (1, 2, 3)

Predicate transitive closure

▪Predicate transitive closure (PTC)

- Optimizer can copy local predicate from one table in join to other table
  - WHERE T1.C1 = T2.C1 AND T1.C1 = ?
  - DB2 will "transitively close" predicate T1.C1 = ? to table T2
    - T2.C1 = ? is added

- Optimizer can copy join predicate from one table in join to other table
  - WHERE T1.C1 = T2.C1 AND T1.C1 = T3.C1
  - Predicate T2.C1 = T3.C1 is added

- Supported local predicate types
  - COL op LIT where op is
    - =, <>, >, <, <=, >=
  - COL (NOT) BETWEEN ? AND ?

Predicate transitive closure

▪ Predicate transitive closure

- ▸ Local predicates

```
SELECT   cols
FROM     T1, T2
WHERE    T1.C1 = T2.C1
AND        T1.C1 = ? ;
```

AND T2.C1 = ?

Can filter on T2 also!

- ▸ Join predicates

```
SELECT   cols
FROM     T1, T2, T3
WHERE    T1.C1 = T2.C1
AND                T2.C1 = T3.C1 ;
```

AND T1.C1 = T3.C1

Can join T1 and T3 directly!

PTC limitations

▪PTC not supported for all predicates.

- IN-LIST, LIKE
- COL IN (NON-CORRELATED SUBQ)
- Compound predicates

Red predicates not currently transitively closed to T2.

```
SELECT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C2 = T2.C2                                   <-- Join
  predicates
AND T1.C1 IN
        (SELECT C1 FROM T3 WHERE T3.Cx = ?)
AND T1.C1 LIKE 'XX%'
AND T1.C1 IN ('A', 'B', 'C')

AND (T1.C1 = ? OR T1.C2 = ?)          <-- Only uses join predicates...
;
```

PTC limitations, user action

- **Why these predicates not transformed**
  - **Like predicate can eliminate previous index only access**
  - **IN-LIST / compound predicates**
    - **Prepare cost to look in compound cases expensive**
    - **Can cause more SQLCODE -101 errors**
  - **COL-IN SUBQUERY**
    - **SUBQUERY would be executed twice**

  - **More research required to extend PTC while avoiding pitfalls**

  - **User action:**
    - **Consider manually adding these predicates**
    - **Will increase optimization oppurtunities**
    - **New oppurtunity may be more efficient path**

Predicate pushdown

Predicate pushdown

- Predicate refers to materialized view / table expression
  - Simple
  - Boolean term
- Pushdown supported predicate types
  - COL op LIT where op is
    - =, <>, >, <, <=, >=
  - (NOT) LIKE
  - (NOT) BETWEEN LIT and LIT
  - COL IS (NOT) NULL
  - IN-LIST
    - V7 APAR PQ73454
    - ZPARM INLISTP controls thus (V7 defaults to off)
    - V8 default allows pushdown

Predicate pushdown example

- Predicate pushdown (cont.)

```
SELECT *
FROM
   (SELECT  REGION, YEAR, QTR, SUM(SALES)
    FROM     SALES_TABLE
    GROUP   BY REGION, YEAR, QTR)  QTR_SALES
WHERE QTR = 1 ;
```

Predicate

```
SELECT *
FROM
   (SELECT   REGION, YEAR, QTR, SUM(SALES)
    FROM      SALES_TABLE
    WHERE QTR = 1
    GROUP BY REGION, YEAR, QTR)  QTR_SALES ;
```

Push inside

Join transformations

- Join transformations
  - Subquery to join transformation
  - Join type reduction

Subquery to join example

▪Subquery to join transformation example

```
SELECT   *
FROM     EMP
WHERE    DEPTNO IN
  ( SELECT  DEPTNO FROM DEPT
    WHERE   LOCATION IN ('TAMPA', 'MIAMI')
       AND  DIVISION = 'MARKETING' ) ;
```

transform

```
SELECT  EMP.*
FROM    EMP, DEPT
WHERE   EMP.DEPTNO = DEPT.DEPTNO
AND   DEPT.LOCATION IN ('TAMPA', 'LA')
AND   DEPT.DIVISION = 'MARKETING'  ;
```

- **Contains**
  - Unique index on (DIVISION, DEPTNO) --> Unique index guarantees no redundancy
  - No local filtering provided on EMP table
- **Benefits**
  - Can consider different join sequences such as DEPT table first using index on division and local filtering on location in-list
  - Can consider different join methods which previously were not available

Join type reduction

- Join type reduction
  - Full outer join transformed to left outer join
    - Full outer join can only use sort merge join
    - Left outer join allows nested loop join also
    - Uni-directional join operation typically more efficient
  - Left / right outer join transformed to inner join
    - Inner join allows all join sequences and join methods
    - Opens up other transformation possibilities

Left to inner join reduction

▪ Left to inner join type reduction example

```
SELECT  *
FROM EMP  E
   LEFT OUTER JOIN  DEPT D
   ON  E.DEPTNO = D.DEPTNO
WHERE  D.DIVISION = 'MARKETING' ;
```

transform

```
SELECT  *
FROM EMP  E
   INNER JOIN  DEPT D
   ON  E.DEPTNO = D.DEPTNO
WHERE  D.DIVISION = 'MARKETING' ;
```

- The where clause predicate filters all the nulls from DEPT table, so DB2 determines the join type can be reduced to inner join

- Benefits
  - Inner join allows alternative join sequences so DEPT table to be outer table. Can use index access via DEPT table
  - Also opens up Hybrid join as possible join method
  - Join type transformation can also reduce materialization

Full to left join reduction

- **Full to left join type reduction example**

```
SELECT  *
FROM EMP  E
   FULL OUTER JOIN  DEPT D
   ON  E.DEPTNO = D.DEPTNO
WHERE  E.EMP_TYPE  <> 'MANAGER' ;
```

transform

```
SELECT  *
FROM EMP  E
   LEFT OUTER JOIN  DEPT D
   ON  E.DEPTNO = D.DEPTNO
WHERE  E.EMP_TYPE  <> 'MANAGER' ;
```

- The where clause predicate filters all the nulls from EMP table, so DB2 determines the join type can be reduced to LEFT OUTER JOIN.

- Benefits
  - LEFT OUTER JOIN allows NESTED LOOP JOIN and SORT MERGE JOIN, FULL OUTER JOIN --> stuck with SMJ
  - Uni-directional sort merge join typically more efficient than bi-directional sort merge join
  - Can cascade to allow more transformations to EMP table

View / table expression merge

- View / table expression merge
  - Allow more join sequences
  - Eliminates expensive materialization
  - Avoid predicate pushdown limitations
  - Allows usage of indexes on base tables
    - Join predicates not pushed into materializations
    - Predicates not pushed down would not be indexable

View merge example

▪View / table expression transformations

Creating view:

```
CREATE VIEW   V1   AS
SELECT  T1.C1, T2.C2
FROM      T1, T2
WHERE   T1.C1 = T2.C1 ;
```

View referencing select:

```
SELECT   V1.*
FROM      V1, T3
WHERE    V1.C2 = T3.C2
    AND    T3.C4 = ? ;
```

Merged result:

```
SELECT   T1.C1, T2.C2
FROM      T1, T2, T3
WHERE    T1.C1 = T2.C1
    AND    T2.C2 = T3.C2
    AND    T3.C4 = ? ;
```

☹ **If view V1 materializes**

- T1 and T2 would be joined unfiltered.
- Possible large workfile
- No index access on workfile
- Fewer join sequences, join methods

☺ **With view merge we have more options**

- Any join sequence
- More indexes available
- More join methods possible

Query Transformation

▪Distribution and Pruning

- Distribution of predicates within union all in view
  - Local predicates
  - Join predicates
  - Aggregates
- Pruning
  - Eliminate query blocks with always false predicates
  - Mostly used to prune always false branches of union all in view designs
  - Also can be used to prune non-union all query blocks

Query Transformation

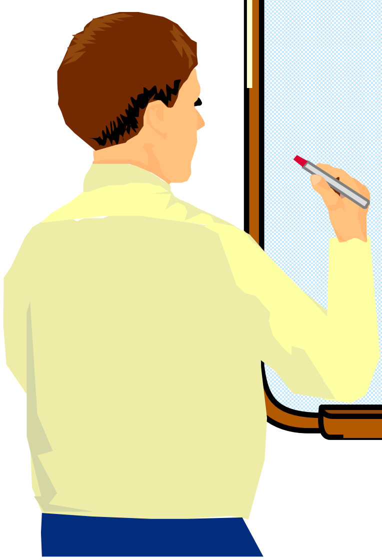
▪Distribution and Pruning (cont'd)

•Distribution and pruning (cont'd)

```
CREATE VIEW transaction( ...... ) AS
SELECT *
FROM first_season
WHERE date BETWEEN '2002-01-01' AND '2002-3-31'
  UNION ALL
SELECT *
FROM second_season
WHERE date BETWEEN '2002-04-01' AND '2002-6-30'
  UNION ALL
SELECT *
FROM third_season
WHERE date BETWEEN '2002-07-01' AND '2002-9-30'
  UNION ALL
SELECT *
FROM fourth_season
WHERE date BETWEEN '2002-10-01' AND '2002-12-31' ;
```

```
SELECT *
FROM transaction T, products P,
        customers C, dates D
WHERE T.pid = P.id     AND
        T.cid = C.id     AND
        T.did = D.id     AND
        C.zipcode IN ( ......)     AND
        T.date BETWEEN '2002-4-15'
                AND '2002-5-15' ;
```

Agenda



Part A

Session 1: Overview

Session 2: Access path and explain table

Session 3: DB2 Runtime Architecture and predicate application

Session 4: Access methods

Part B

Session 5: Join methods

Session 6: Query transformation

Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

Statistics

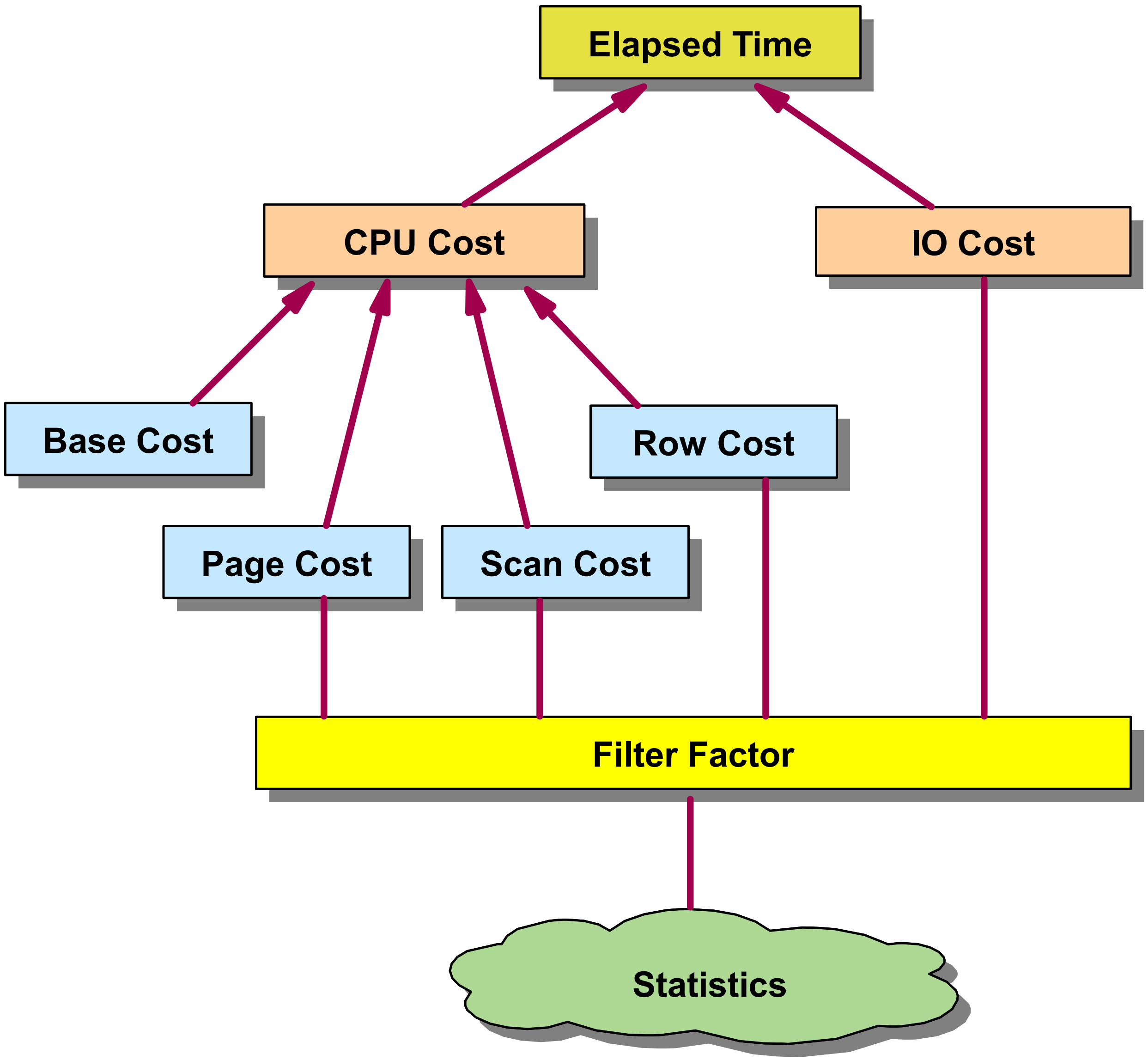

Elapsed Time
CPU Cost
IO Cost
Base Cost
Row Cost
Page Cost
Scan Cost
Filter Factor
Statistics

Table Statistics

- CARDF
  - Number of rows in a partition/table
- NACTIVEF
  - Number of active pages for table space
  - Only used for single table simple tablespaces
- NPAGESF
  - Number of pages where rows appear in a partition/table
- PCTROWCOMP
  - Percentage of compressed rows

Index statistics

- NLEAF
  - Number of active leaf pages
- NLEVELS
  - Number of levels in the index tree
- CLUSTERRATIOF
  - Percentage of rows in clustering order

Selectivity statistics

- Single column
  - Cardinality
  - HIGH2KEY/LOW2KEY
  - Frequency
- Multi-column
  - Cardinality
  - Frequency

Single column cardinality

- Single column cardinality
  - Number of distinct values for a column
  - Assumes uniform distribution
  - Stored as
    - SYSCOLUMNS.COLCARDF
    - SYSINDEXES.FIRSTKEYCARDF
  - Used when better statistics can't be used...
    - Host variables, parameter markers, special registers
    - No other statistics available

HIGH2KEY/LOW2KEY

- **HIGH2KEY/LOW2KEY**
  - **Single column statistic**
    - **SYSCOLUMNS.HIGH2KEY**
    - **SYSCOLUMNS.LOW2KEY**
  - **When used?**
    - **Interpolation used to estimate range predicates**
    - **Like, between, <, <=, >, >=**
    - **Literal value must be known**
    - **As domain statistics when COLCARDF = 1 or 2**
    - **Can be used in combination with single column frequencies for more accurate estimate.**
    - **DB2 Interpolation: Technique to estimate the percentage of rows which qualify based on known high / low values.**

Single column frequencies

- Single column frequencies
  - SYSCOLDIST.FREQUENCYF
    - TYPE = 'F', NUMCOLUMNS = 1
  - Provides non-uniform distribution information
    - Data skew
  - When used?
    - Literal value must be known
    - Equals, is null, in
    - Like, between, <, <=, >, >=
    - Used in conjunction with other complementary statistics

Multi-column cardinalities

- Multi-column cardinalities (MCARD)
  - Stored in a few places...
    - SYSINDEXES.FULLKEYCARDF
    - SYSCOLDIST.CARDF
      - TYPE = 'C', NUMCOLUMNS > 1
  - Assumes uniform distribution
  - When used?
    - Primarily for indexes
    - Literal values not necessary
    - KEYCARD for partially matching indexes
      - Collect for all indexes with 3 or more columns
    - Collect to support multi-column frequencies
    - Collect for all multi-column join situations

Multi-column frequency

- **Multi-column frequencies**
  - **Very similar to single column frequencies**
    - **Distribution statistics concatenated column group values**
    - **Identifies multi-column skewed distributions**
  - **Stored in**
    - **SYSCOLDIST.FREQUENCYF**
    - **TYPE = ‘F’**
    - **NUMCOLUMNS > 1**

Multi-column frequency

- Multi-column frequencies
  - Limited use
    - Boolean equal predicates only
    - Always collect supporting multi-column cardinality
  - Collect single column frequencies for
    - Range predicates
    - In-lists
    - Single column predicates
    - other non-equal predicates

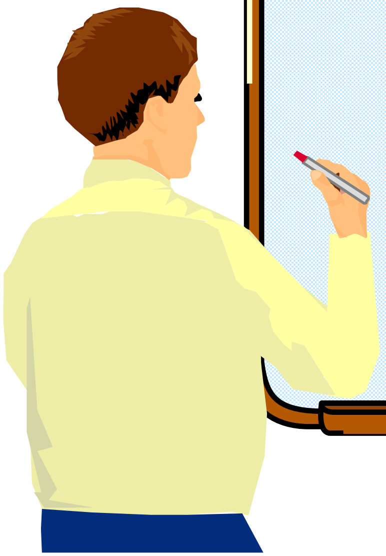
Statistics advisor

- Problem: Manual predicate analysis is time consuming and error prone

- Proposal
  - Automate much of the analysis
    - Identify predicates using default statistics
    - Identify statistics inconsistencies
    - Identify predicates with questionable filter factor
    - Identify probable correlation situations
  - Generate appropriate RUNSTATS commands

- DB2 Statistics Advisor is generally available as part of DB2 V8 Visual Explain in September, 2004

Agenda



Part A

Session 1: Overview

Session 2: Access path and explain table

Session 3: DB2 Runtime Architecture and predicate application

Session 4: Access methods

Part B

Session 5: Join methods

Session 6: Query transformation

Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

- **Patrick Bossman**
  - **Z32: Don’t miss the overhauled DB2 for z/OS Visual Explain V8**
  - **Z33: Control your own destiny – Implementing DB2 for z/OS Optimization hint**
- **Terry Purcell**
  - **Z34: DB2 for z/OS Exploiting the V7 & V8 Optimization enhancements – Part A**
  - **Z34: DB2 for z/OS Exploiting the V7 & V8 Optimization enhancements – Part B**