Z30 – Part A

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

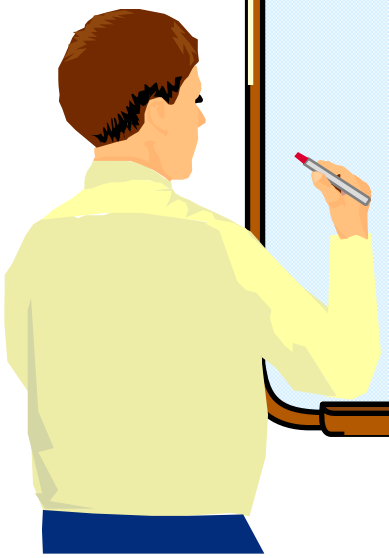
Session 5: Join methods

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

What is optimization

- What is optimization?

For a given SQL statement, select the access path which returns the correct result with minimum elapsed time.

Determine access path

- SQL is declarative language
  - SQL tells database WHAT not HOW
    - What information should database return
    - Not how to get the information
  - In general, more than one way to evaluate query

- Different from procedural languages
  - Eg. C, COBOL, REXX
  - Define how the information should be processed

A sample SQL query

- Query

```
select d.name, sum(e.id), avg(e.compensation)
from employees e, departments d, projects p
where d.location in ('SVL', 'ARC')
    and d.id = e.dept
    and e.id = p.employeeID
    and p.class in ('top secret', 'confidential')
group by d.name
order by d.name
```

Aggregate functions

(Inner) joins

Local predicates (P1 and P4)

Join predicates (P2 and P3)

Aggregate (grouping)

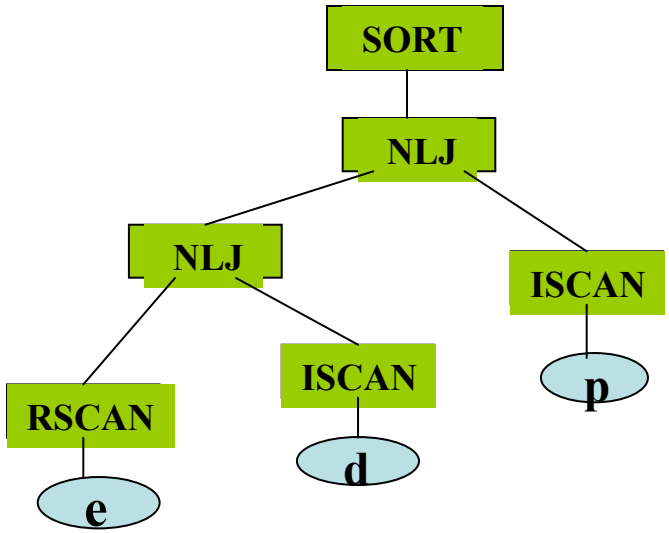
Ordering the result set

- Assumptions
  - Index: ix1(d.location), ix2(e.id), ix3(e.dept), ix4(p.class)
  - Cardinality: card(e) = 300000, card(d) = 20000, card(p) = 50000
  - Filtering factor: FF(P1) = 1%, FF(P4) = 10%

A sample SQL query

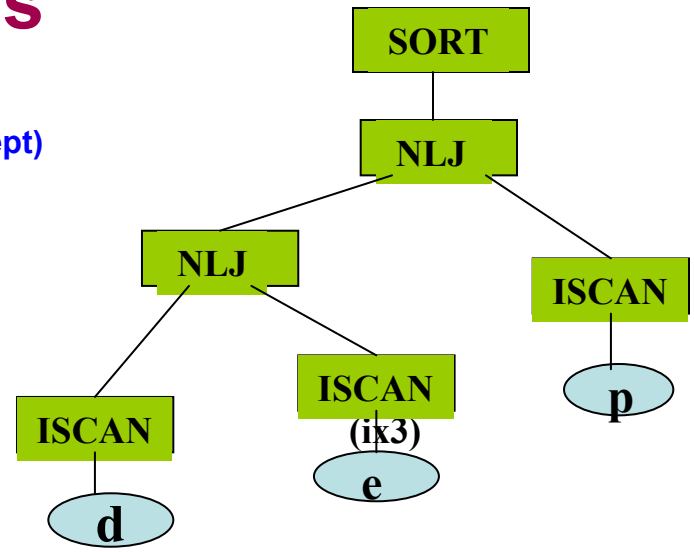
Query

```
select d.name, sum(e.id), avg(e.compensation)
from employees e, departments d, projects p
where d.location in ('SVL', 'ARC')
    and d.id = e.dept
    and e.id = p.employeeID
    and p.class in ('top secret', 'confidential')
group by d.name
order by d.name
```



Assumptions

- Index: ix1(d.location)
  ix2(e.id), ix3(e.dept)
  ix4(p.class)
- Cardinality:
  card(e) = 300000
  card(d) = 20000
  card(p) = 50000
- Filtering factor:
  FF(P1) = 1%,
  FF(P4) = 10%



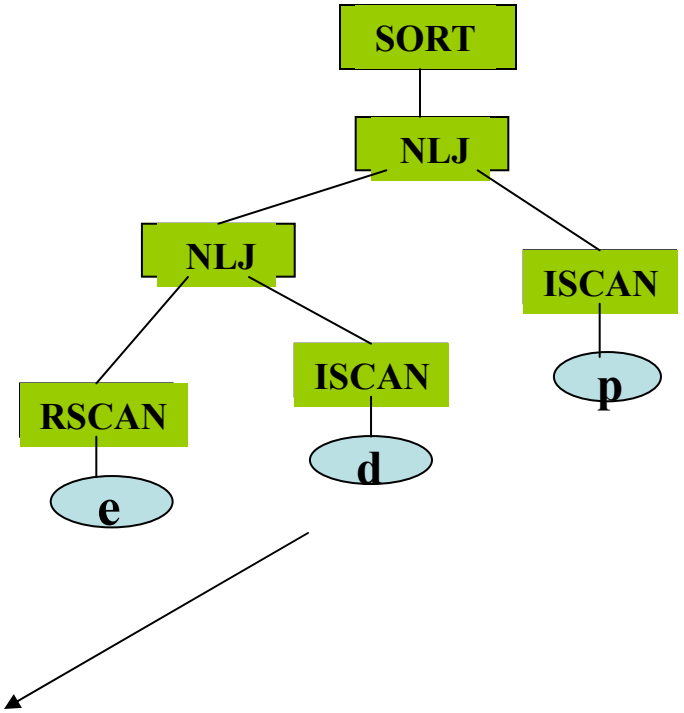
Access Path

A sample SQL query

Query

```
select d.name, sum(e.id), avg(e.compensation)
from employees e, departments d, projects p
where d.location in ('SVL', 'ARC')
    and d.id = e.dept
    and e.id = p.employeeID
    and p.class in ('top secret', 'confidential')
group by d.name
order by d.name
```

Access Path



Assumptions

- Index: ix1(d.location)
  ix2(e.id), ix3(e.dept)
  ix4(p.class)
- Cardinality:
  card(e) = 300000
  card(d) = 20000
  card(p) = 50000
- Filtering factor:
  FF(P1) = 1%,
  FF(P4) = 10%

- Number of join sequences (3! = 6)
- Number of access methods (RSCAN, ISCAN, etc)
- Number of join methods (NLJ, SMJ, HBJ, SJ)
- Total number of access paths
- Cost estimation (elapsed time)
- Access path selection (shortest elapsed time)

A sample SQL query – join transformation

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

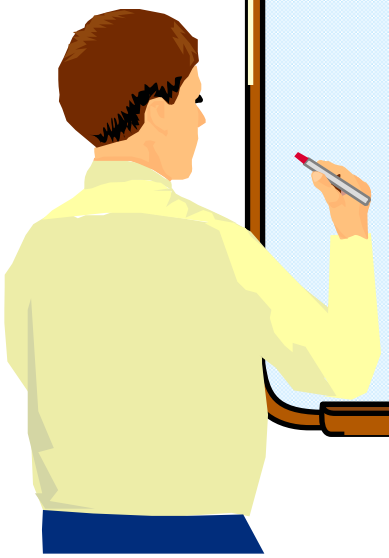

```
SELECT   EMP.*
FROM     EMP, DEPT
WHERE    EMP.DEPTNO = DEPT.DEPTNO
AND    DEPT.LOCATION IN ('TAMPA', 'LA')
AND    DEPT.DIVISION = 'MARKETING'  ;
```

- 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 previosly were not available

Overview of DB2 Optimizer

- Query transformation (cost independent)
- Access path enumeration
- Cost estimation
- Parallelism optimization (not covered in this presentation)
- Explain of optimal access path

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

Single Table Access Method Execution

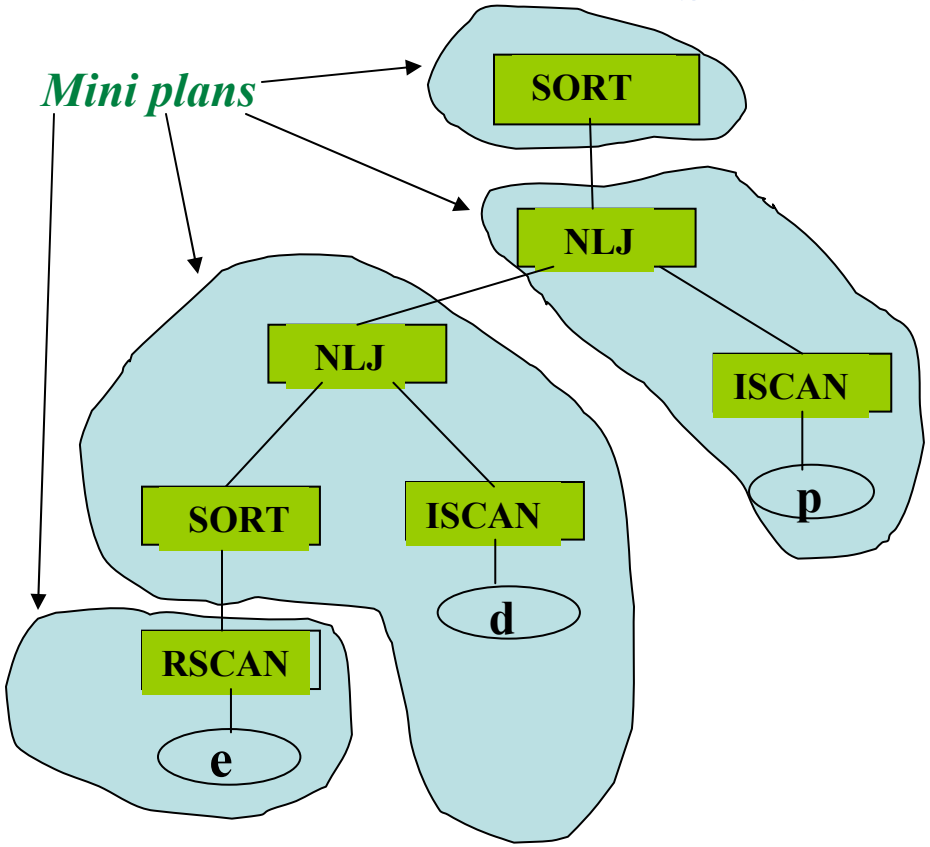
- This section:
  - Objectives
    - To introduce DB2 PLAN_TABLE.
  - DB2 PLAN_TABLE
    - Mini plan
    - PLAN_TABLE columns

Mini plan

- Query

```
select d.name, sum(e.id), avg(e.compensation)
from employees e, departments d, projects p
where d.location in ('SVL', 'ARC', 'Toronto', 'Raleigh')
    and d.id = e.dept
    and e.id = p.employeeID
    and p.class in ('top secret', 'confidential')
group by d.name
```

- Mini plan
  - Plan record identification
  - Access of new table
  - Sort of new table
  - Join method
  - Sort of the composite



Plan Table Columns – Plan record identification

VERSION	Version identifier of a package - in embedded SQL
COLLID	Collection id for a package. VARCHAR(128) in V8.
APPLNAME	Name of application plan for static SQL
PROGNAME	Name of the program or package that contains the statement. VARCHAR(128) in V8.
QUERYNO	A number intended to identify the query being explained
QBLOCKNO	A number indicating a query or subquery block, showing the block's order in the SQL. Subqueries can be merged into one block by the optimizer.
PLANNO	Processing sequence of the step within QBLOCKNO; each new table accessed has a new step in the plan.
TIMESTAMP	When the EXPLAIN was executed

Plan Table Columns – Access of new table (1/4)

CREATOR	Creator of the table accessed in the plan step. VARCHAR(128) in V8.
TNAME	Name of a table, created or declared temporary table, materialized view, table expression or outer join workfile accessed in this step. VARCHAR(128) in V8.
TABNO	Identifies the FROM clause table showing the position of reference in the SQL
ACCESSTYPE	How the table is accessed
I	Index scan
I1	One-fetch index scan
M	Multi-index access. Always followed by MX, MI or MU
MX	Multi-index scan on referenced index
MI	Intersection of multiple indexes
MU	Union of multiple indexes
N	IN list index scan
R	Tablespace scan (Relational Scan)
RW	Workfile scan of the result of a materialized user-defined table function
T	Sparse Index or In-memory Workfile Access
V	Buffers for an INSERT statement within a SELECT
blank	Not applicable to this row

Plan Table Columns - Access of new table (2/4)

CORRELATION_NAME	Correlation name of the view/table specified in the statement. VARCHAR(128) in V8.
PAGE_RANGE	Whether the table qualifies for page range screening
TABLE_TYPE	The type of new table
B	Buffers for INSERT statement within a SELECT
C	Common Table Expression
F	Table function
M	Materialized Query Table
Q	Non-materialized temporary intermediate result table
RB	Recursive Common Table Expression
T	Table
W	Work file
NULL	Implicit sort for GROUP BY, ORDER BY, or DISTINCT

Plan Table Columns - Access of new table (3/4)

PRIMARY_ACCESS_TYPE	Indicates whether direct row access will be attempted
D	DB2 will try to use direct row access
blank	DB2 will not try to use direct row access
TSLOCKMODE	Show the tablespace lock mode LOCK is IS, IX, S, U, X, SIX, N (NS, NIS, NISS, SS)
PREFETCH	Shows which form of Prefetch is used
S	Sequential Prefetch
L	List Prefetch
D	Optimizer cost assumes Dynamic Prefetch at runtime
blank	No prefetch or unknown
COLUMN_FN_EVAL	Shows when an SQL column function was evaluated
R	At data retrieval time
S	At sort time
blank	At data manipulation or unknown

Plan Table Columns - Access of new table (4/4)

ACCESSTYPE	How the table is accessed
I	Index scan
I1	One-fetch index scan
M	Multi-index access. Always followed by MX, MI or MU
MX	Multi-index scan on referenced index
MI	Intersection of multiple indexes
MU	Union of multiple indexes
N	IN list index scan
R	Tablespace scan (Relational Scan)
RW	Workfile scan of the result of a materialized user-defined table function
T	Sparse Index or In-memory Workfile Access
V	Buffers for an INSERT statement within a SELECT
blank	Not applicable to this row
ACCESSCREATOR	Index Creator if ACCESSTYPE is I, I1, N, or MX. VARCHAR(128) in V8.
ACCESSNAME	Index Name if ACCESSTYPE is I, I1, N, MX. VARCHAR(128) in V8.
INDEXONLY	Y/N for index access with no data reference (write to data pages for UPDATE is ignored by this flag)
MATCHCOLS	Number of matched columns in the INDEX key where ACCESSTYPE is I, I1, N, or MX

Plan Table Columns – Join method

METHOD	Number showing the join method used in the plan
0	First table accessed, continuation of previous table – or not used
1	Nested loop join
2	Sort Merge (Merge scan) join
3	Additional sorts for ORDER BY, GROUP BY, DISTINCT, UNION etc.
4	Hybrid join
MERGE_JOIN_COLS	Number of columns joined using a Merge Scan Join (Method=2)
JOIN_TYPE	The type of join
F	Full Outer Join
L	Left Outer Join (Optimizer converts Right Joins to Left Joins)
S	Star Join
blank	Inner Join or no join

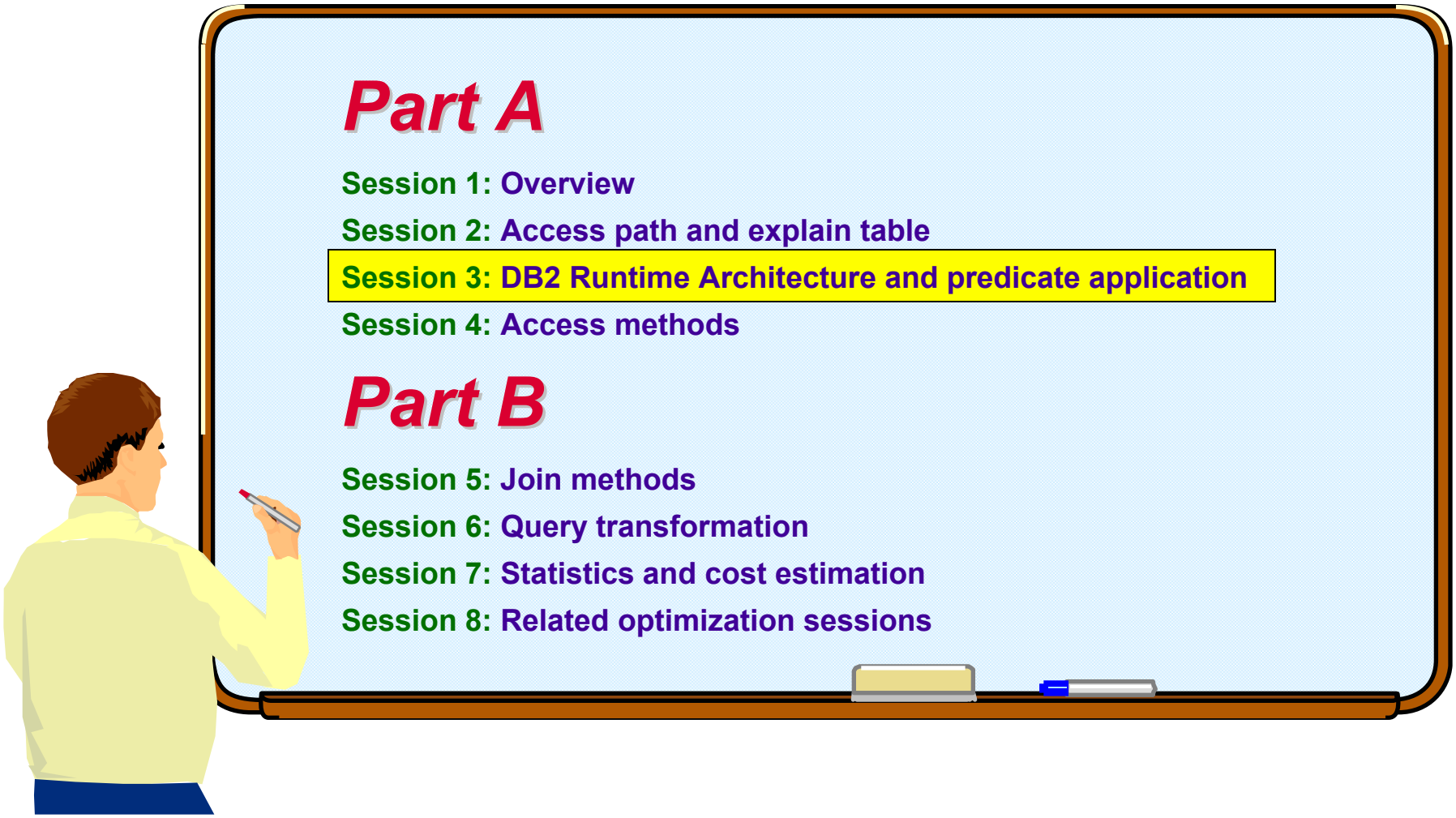
Plan Table Columns – Sorting new table

SORTN_UNIQ	Sort on new (inner) table to remove duplicates (not used)
SORTN_JOIN	Sort on new (inner) table for join method 2 or 4. Only valid for method 1 during outside in phase of star join.
SORTN_ORDERBY	Sort on new (inner) table for an ORDER BY (not used)
SORTN_GROUPBY	Sort on new (inner) table for a GROUP BY (not used)

Plan Table Columns - Sorting composite

SORTC_UNIQ	Sort on composite (outer) table to remove duplicates
SORTC_JOIN	Sort on composite (outer) table for a join method 1, 2 or 4
SORTC_ORDERBY	Sort on composite (outer) table for an ORDER BY or a quantified predicate
SORTC_GROUPBY	Sort on composite (outer) table for a GROUP BY

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

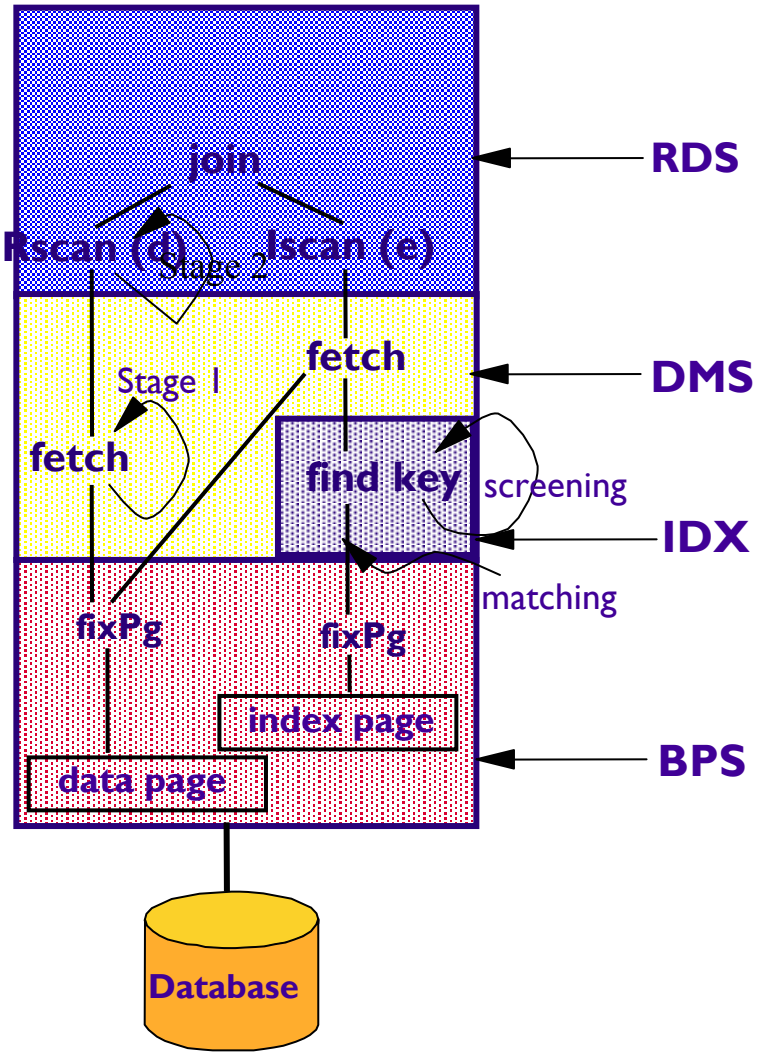
Session 7: Statistics and cost estimation

Session 8: Related optimization sessions

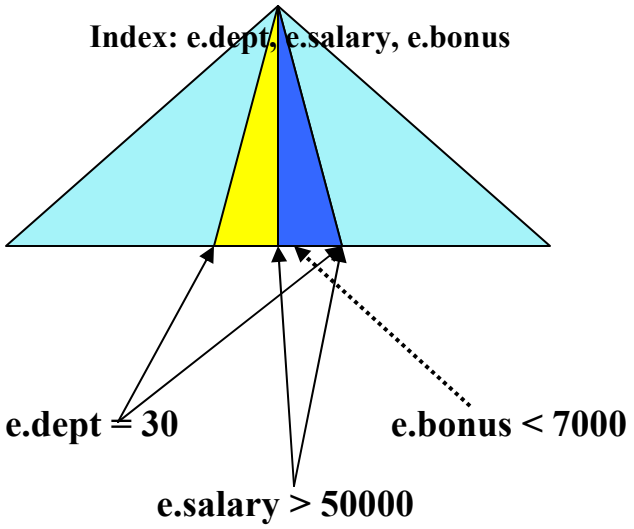
Predicate Sargability

- **This section:**
  - Objectives
    - To obtain an understanding of the different predicate types and differentiate the sargability (or stage of application) of SQL predicates.
  - Introduces
    - Predicate Properties
    - Predicate Types
    - Predicate Application Stages
    - Indexability
    - Improving predicate performance

DB2 run-time architecture




```
select * from dept d, emp e
 where d.budget > 10000          // Stage 1
   and d.name like '%Opt%'       // Stage 2
   and d.id = e.dept             // Matching
   and e.salary > 50000          // Matching
   and e.bonus < 7000            // Screening
```



Matching Predicates

- **Index Matching Predicates:**
  - Restrict the range of data that is retrieved
    - All other predicate types will subsequently reject rows based upon this retrieved range of data
  - Only indexable predicates can be matching
  - Number of matching predicates depends on the chosen index
    - Matching predicates on the leading index columns are generally '=' or IN.
    - Once a range predicate is encountered (or 2 IN predicates), subsequent index columns are not considered matching

Index 1: Firstnme, Lastname Index 2: Lastname, Firstnme, City

1 matching column

```
WHERE LASTNAME = 'SMITH'
AND   FIRSTNME LIKE 'J%'
AND   CITY = 'CHICAGO'
```

2 matching columns

Index Screening

- Stage 1 predicates not chosen as matching may be applied on the index before data access:
  - Referred to as Index Screening
  - Generally restricted to simple stage 1 predicates
  - Provided the column exists in the chosen index
  - Predicate may be index screening and not matching because
    - The predicate is stage 1 but not indexable
    - Or the predicate is indexable, but cannot match on an index
      - **One or more preceding index columns are not matching or are missing**
      - **Or, one or more preceding index columns is a range predicate**
      - **Or, of the leading index columns, there exists more than one IN list**
  - Index screening predicates do not limit the number of index of index entries read
    - But can limit the number of data rows retrieved

Index Screening

- **Index Screening examples:**
  - Assume an index on
    - WORKDEPT, LASTNAME, FIRSTNME

1. `WHERE LASTNAME = 'SMITH'`

Index screening. No match on leading column.

2. `WHERE WORKDEPT = 'A00'`
   `AND FIRSTNME = 'JOHN'`

Screening. Missing a match on LASTNAME

3. `WHERE WORKDEPT = 'A00'`
   `AND LASTNAME LIKE 'SMI%'`
   `AND FIRSTNME = 'JOHN'`

Screening. Following a range predicate

Predicate Application

- **Stage 1 and Stage 2 Predicates:**
  - Stage 1 Predicates
    - Sometimes referred to as Sargable
    - Can be applied at the 1st stage of predicate processing
    - All indexable predicates are also stage 1
      - **But not all stage 1 predicates are indexable**
  - Stage 2 Predicates
    - Sometimes referred to as Nonsargable or Residual
    - Cannot be applied until the 2nd stage of predicate processing
      - **And are therefore not indexable**
  - The following may determine whether a predicate is stage 1 or 2
    - Predicate syntax (see following tables for examples)
    - Type and length of constants or columns in the predicate
      - **DB2 V8 resolves most of these**
    - Whether the predicate is applied before or after a join
    - Table join sequence

Indexable Predicates

COL op value	op is =, >, >=, <, <=
COL op noncol-expr	
COL IS NULL	
COL IS NOT NULL	
COL BETWEEN value1 AND value2	
COL BETWEEN expr1 AND expr2	column expr has a join sequence dependency
COL LIKE 'pattern'	'pattern' cannot begin with wildcards % or _
COL LIKE host-variable	same rules as 'pattern'
COL IN (list)	must only contain constants, host variables, parameter markers or special registers
T1.COL op T2.COL	
T1.COL op T2 col-expr	Join sequence dependency
COL op (noncorrelated subquery)	
COL IN (noncorrelated subquery)	
(COL1,COL2,.....) IN (noncorrelated subquery)	

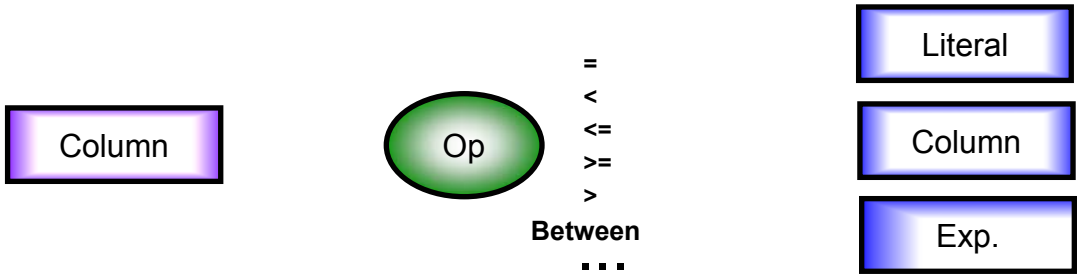
Stage 1 Predicates

COL <> value	
COL <> noncol-expr	
COL NOT BETWEEN value1 AND value2	
COL NOT BETWEEN noncol expr1 AND noncol expr2	
COL NOT IN (list)	
COL NOT LIKE 'pattern'	
COL LIKE '%pattern'	or '_pattern' - wildcard 1st char
T1.COL <> T2 col-expr	
COL <> (noncorrelated subquery)	

Stage 2 Predicates

value (NOT) BETWEEN col1 AND col2	Value between two columns
T1.COL1 op T1.COL2	Two columns from the same table
T1.COL1 <> T1.COL2	
COL op ALL (subquery)	Subquery = Correlated or Non-correlated
COL op ANY (subquery)	
COL <> ALL (subquery)	
COL <> ANY (subquery)	
COL NOT IN (noncorrelated subquery)	
COL op (correlated subquery)	
COL <> (correlated subquery)	
COL (NOT) IN (correlated subquery)	
(NOT) EXISTS (subquery)	
expr op value	Two values/expressions/host variables compared. NOTE: May be pruned if invalid.
expr <> value	
expr op (subquery)	Value/expression compared to a subquery

Datatype/Length – V8 and prior



Up through V7

Same type, length, ccsid — **Same type, length, ccsid**

In V8

All except ...

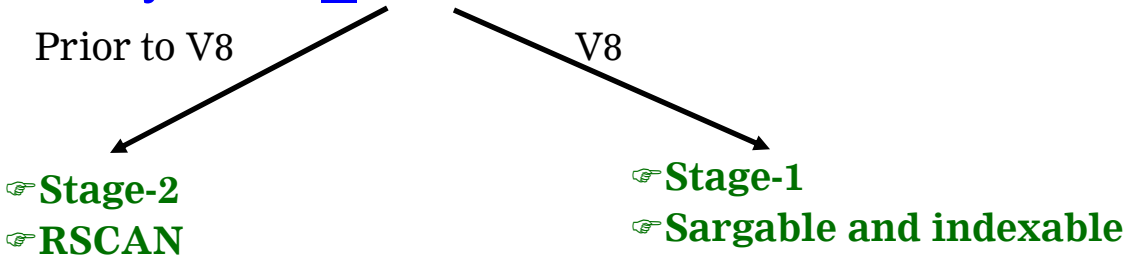
Dec(p,s), where p > 15	**Float**
string types	**Date, Time or Timestamp**
Unicode or same CCSID	**Unicode or same CCSID**

Datatype/Length mismatch – V8

- Unmatched data type: numeric types

```
Employee ( Name     character (20),
           Salary   decimal (12,2),
           deptID   character (3) );
```

```
SELECT * FROM employee
    WHERE salary > :hv_float ;
```



Datatype/Length mismatch – V8

- Unmatched types: string types

```
SELECT * FROM employee
    WHERE deptID = '6S5A' ;
```

or

```
SELECT * FROM employee
    WHERE deptID = '6S5  ' ;
```

char(3)

Prior to V8

- ☞ **Stage-2**
- ☞ **RSCAN**

V8

- ☞ **Stage-1**
- ☞ **Sargable and indexable**

Join Dependent Indexability in V8

- Unknown join sequence: Column-expression
  - Without datatype/length match

```
SELECT e1.*
    FROM emp AS e1, emp AS e2, dept
    WHERE e1.deptID = dept.id   AND
          dept.mgr = e2.name  AND
          e1.salary > e2.salary * 1.10 ;
```

dec(12,2)

prior to v8

☞ **Stage-2 predicate**

v8

☞ **If e2 is inner table**
- **Stage-2 predicate**

☞ **If e1 is inner table**
- **Stage-1 predicate**
- **Could use index on emp.salary**

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

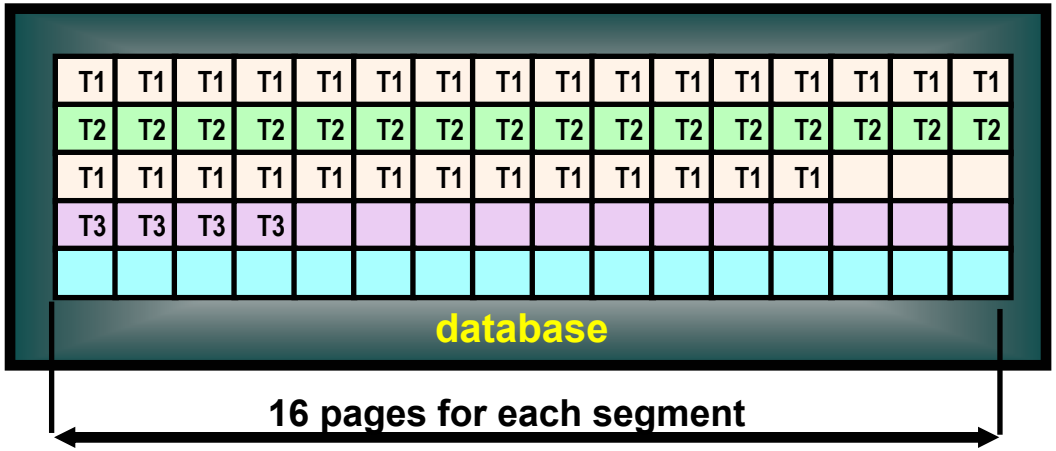
Session 8: Related optimization sessions

Single Table Access Method Execution

- This section:
  - Objectives
    - To highlight the different access methods available for single table access.
  - Single Table Access Methods
    - Tablespace Scan
    - Index Access
    - List Prefetch

Tablespace Scan

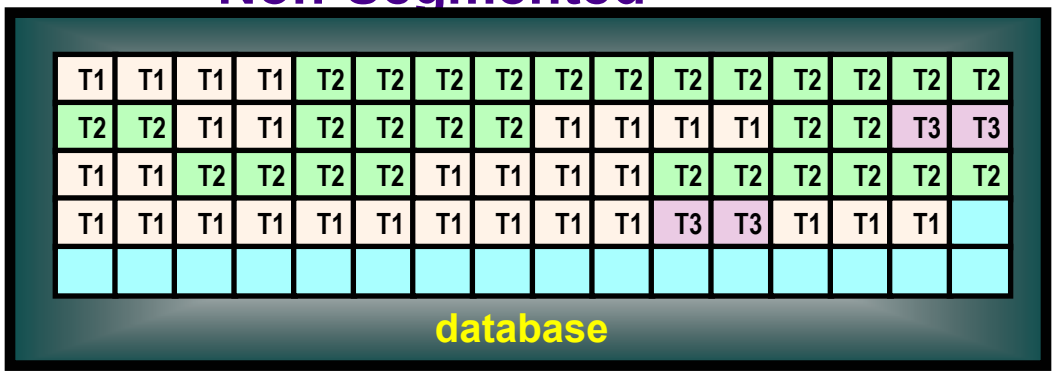
- **Table space scan**
  - **Segmented**




```
CREATE TABLESPACE tablespace
IN database SEGSIZE 16  ..... ;
```

```
SELECT*
FROM        T3
WHERE ..... ;
```

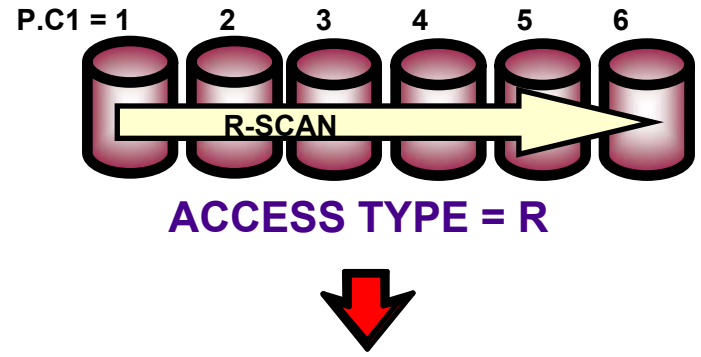
  - **Non-Segmented**




```
CREATE TABLESPACE tablespace
IN database  ..... ;
```

Page Range Screening

- Table space scan
  - Page Range Screening
    - **Also known as Partition Elimination or Limited Partition Scan**




```
SELECT*
FROM    P
WHERE   P.C1 <= 2
AND     ..... ;
```

Page Range column of plan table indicates a subset of the partitions are R-scanned

Sequential Prefetch

- **Sequential Prefetch**
  - Reads a sequential set of pages into the bufferpool with one asynchronous I/O
  - Usually the max number of pages is 32 for base table and 8 for workfile (when VPSIZE => 1000)
  - Generally used for table space scan
  - Sometimes used for index scan when
    - Index clusterratiof > 0.8
    - For index only: number of qualified leaf pages > 8
    - For index + data: number of qualified clustered data pages > 8

Tablespace Scan

```
EXPLAIN PLAN SET QUERYNO = 1 FOR
SELECT * FROM DSN8710.EMP;

EXPLAIN PLAN SET QUERYNO = 2 FOR
SELECT * FROM DSN8710.EMP
OPTIMIZE FOR 1 ROWS;

EXPLAIN PLAN SET QUERYNO = 3 FOR
SELECT * FROM DSN8710.EMP
FETCH FIRST 1 ROWS ONLY;
```

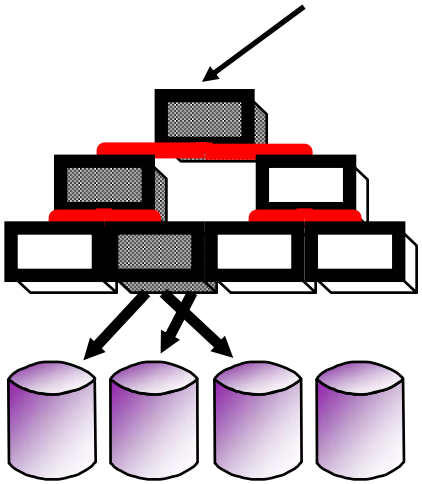
Optimize or Fetch First

QQP	MTH	TNAME	ATYP	MCOL	A_NM	IXO	SORT NCCCC J UJOG	PF	QB_TYP	PQB	TB_TYP
01-01-01	0	EMP	**R**	0		N		**S**	SELECT	0	T
02-01-01	0	EMP	**R**	0		N			SELECT	0	T
03-01-01	0	EMP	**R**	0		N			SELECT	0	T

Sequential Prefetch Impact

Index Scan

- **Index scan (index only / index + data)**
  - **Matching**



```
SELECT *
FROM DSN8710.EMP
WHERE  WORKDEPT = ?
AND       EMPNO = ?
```

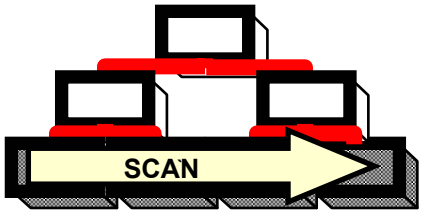
Index + data access

QQP	MTH	TNAME	ATY	MCO	A_NM	IXO	SORT NCCCC J UJOG	PF	MIX	QB_TYP	PQB	TB_TYP
01-01-01	0	EMP	I	2	EMPX2	N			0	SELECT	0	T

Index access with 2 matching columns

Index Scan

- **Index scan (index only / index + data)**
  - **Non-matching**




```
SELECT WORKDEPT
FROM DSN8710.EMP
WHERE  EMPNO > ?
ORDER BY WORKDEPT
```

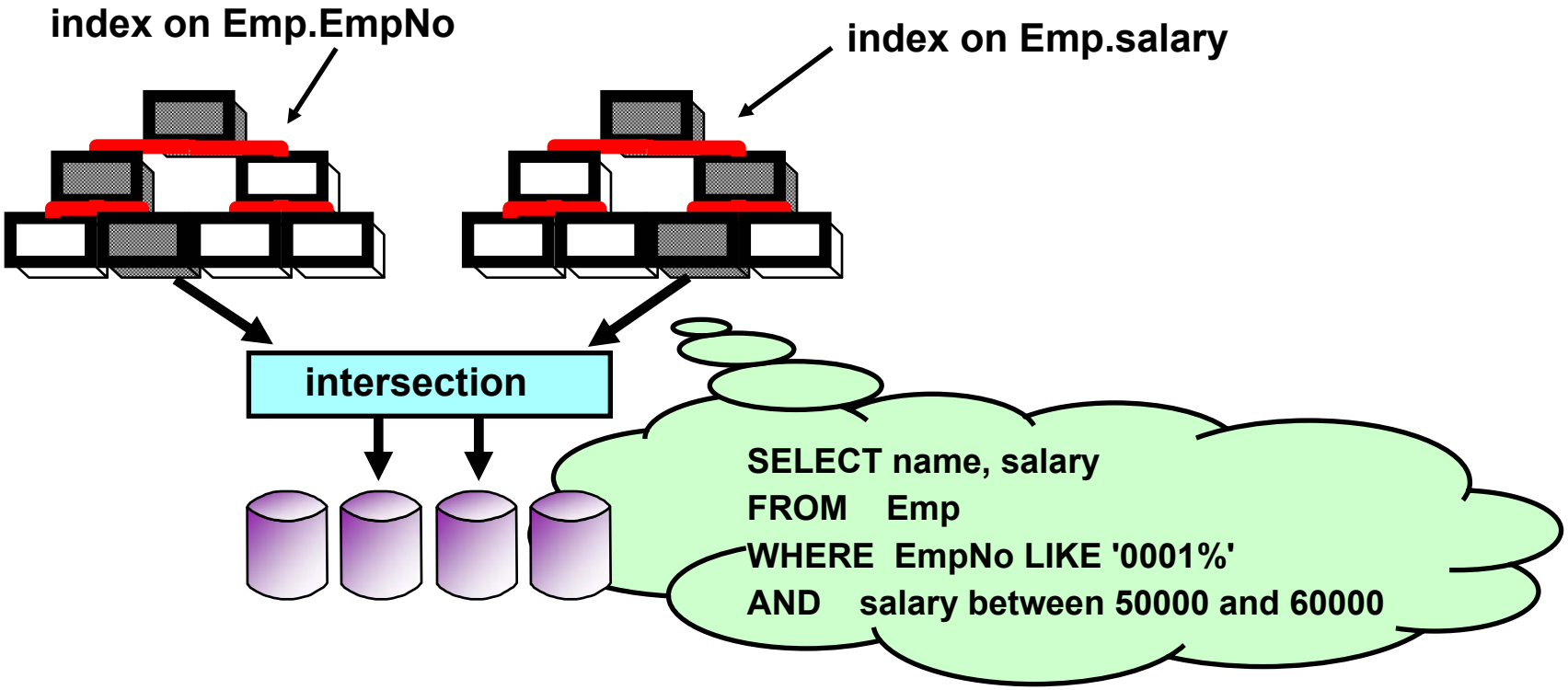
Data retrieval satisfied by index only. No access to table required.

QQP	MTH	TNAME	ATY	MCO	A_NM	IXO	SORTN CCCC J UJOG	PF	MIX	QB_TYP	PQB	TB_TYP
01-01-01	0	EMP	I	0	EMPX2	Y			0	SELECT	0	T

Index access with zero matching columns

Multi-Index Access

- Index scan (index only / index + data)
  - Multi-index access



Multi-Index Access - Example

```
SELECT EMPNO, WORKDEPT, SALARY
FROM DSN8710.EMP
WHERE  (WORKDEPT = ? AND EMPNO = ?)
OR     (WORKDEPT = ? AND EMPNO = ?)
OR     (WORKDEPT = ? AND EMPNO = ?)
ORDER BY SALARY
```

QQP	MTH	TNAME	ATYP	MCOL	A_NM	IXO	SORT NCCCC J UJOG	PF	MIX	QB_TYP	PQB	TB_TYP
01-01-01	0	EMP	M	0		N		L	0	SELECT	0	T
01-01-01	0	EMP	MX	2	EMPX2	Y			1	SELECT	0	T
01-01-01	0	EMP	MX	2	EMPX2	Y			2	SELECT	0	T
01-01-01	0	EMP	MU	0		N			3	SELECT	0	T
01-01-01	0	EMP	MX	2	EMPX2	Y			4	SELECT	0	T
01-01-01	0	EMP	MU	0		N			5	SELECT	0	T
01-01-02	3			0		N	NNNYN		0	SELECT	0	

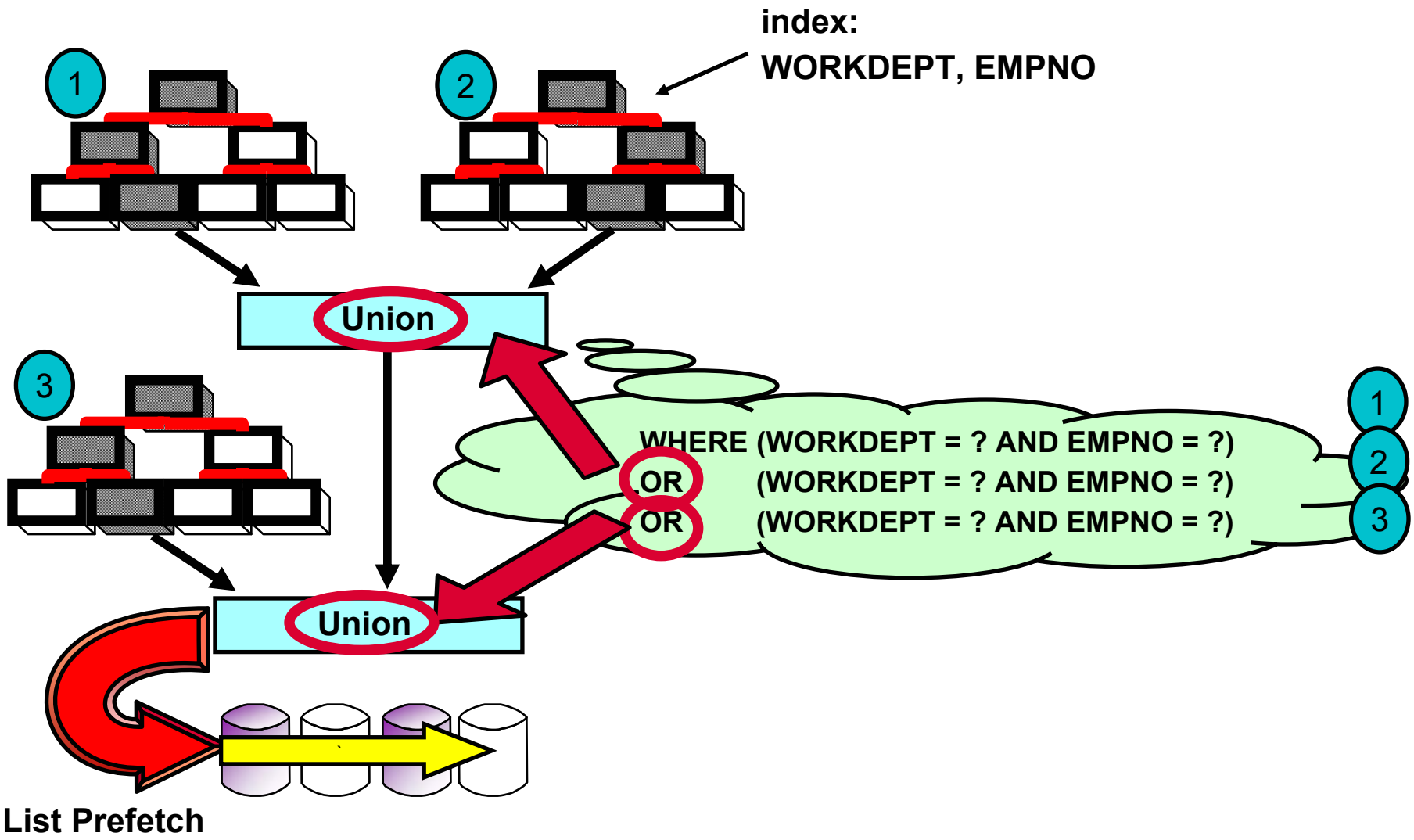
Data Access using List Prefetch

Union output from step 1 & 2

Union output from step 3 & 4

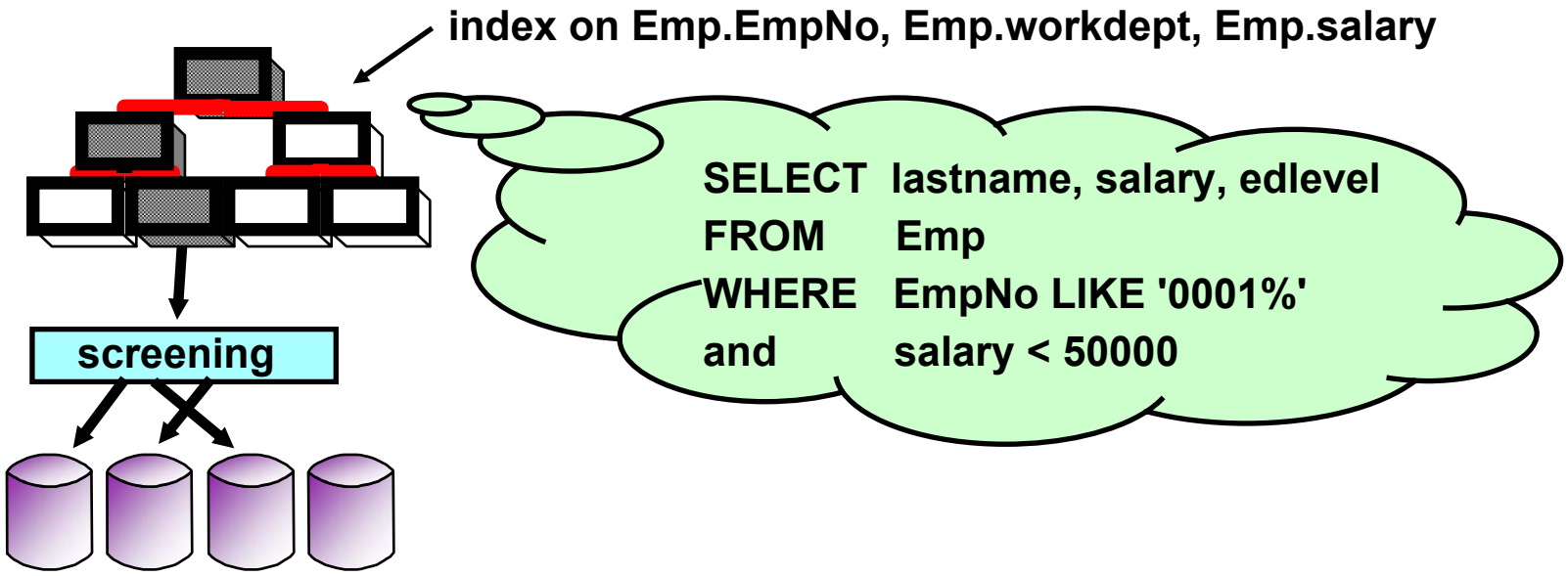
Multi-Index Access Steps

Multi-Index Access - Example



Index Scan - Screening

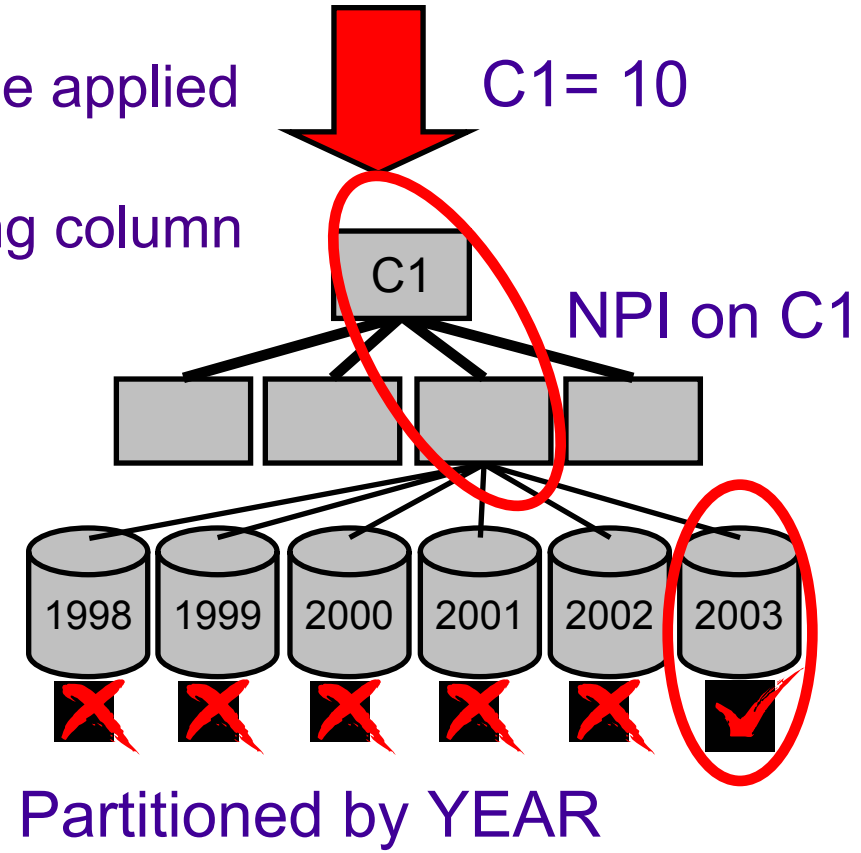
- **Index scan (index only / index + data)**
  - **Index screening**



MATCHCOLS = 1 --> index pages searched on Emp.EmpNo --> performing screening on Emp.salary --> fetch data pages

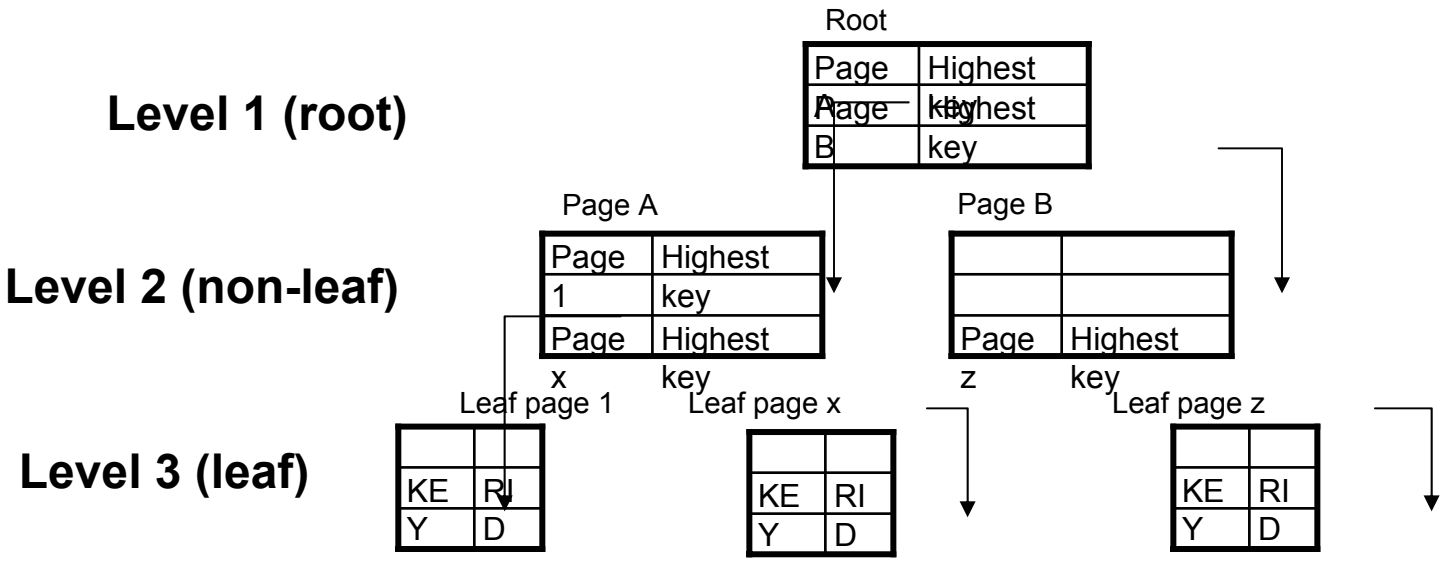
Page Range Screening - NPI

- **Page Range Screening can be applied**
  - before data access on a NPI to limit the partitions accessed
    - if a predicate exists that can be applied
  - Similar to index screening
    - Without requiring the screening column to be indexed




```
SELECT cols
FROM T1
WHERE C1 = 10
AND YEAR = 2003
```

Index Lookaside



- **Objective is to minimize index getpage operations**
  - DB2 checks whether the required entry is in the leaf page accessed by the previous call
    - Check against the low & high key of leaf page
  - If found, getpage is avoided
    - No index tree traversal is required

Index Lookaside

- **Continued**
  - If index key is not within the cached range
    - Check the parent non-leaf page low & high key
  - If found within the parent non-leaf range
    - Get corresponding leaf page
      - Full tree traversal avoided
  - If not found within the parent non-leaf range
    - Must probe index starting from the root page

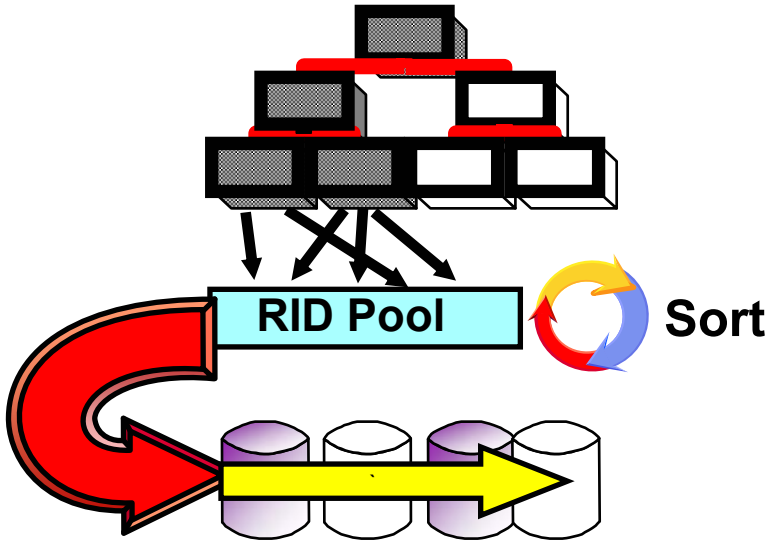
- **Beneficial for repeat index access in sequence**
  - Inner table of nested loop or hybrid join
  - SQL statement within a program loop

List Prefetch

- **List Prefetch concepts**
  - Reads set of pages into bufferpool with one asynchronous I/O
  - The set of pages are determined by a list of RIDs taken from an index
  - Currently the max number of pages prefetched is 32 within 180 page swath (list prefetch can skip pages)
  - Generally used with
    - Index scan when clusterratiof is less than 0.8
    - Accessing data from the inner table during a hybrid join
    - Multi-index access
    - When direct access not possible (for update of...)
    - With high clusterratiof if number of qualified pages between 1 and sequential prefetch

List Prefetch

```
SELECT EMPNO, SALARY
FROM DSN8710.EMP
WHERE  WORKDEPT = ?
ORDER BY SALARY
```



QQP	MTH	TNAME	ATYP	MCOL	A_NM	IXO	SORT NCCCC J UJOG	PF	QB_TYP	PQB	TB_TYP
01-01-01	0	EMP	I	0	EMPX2	N	NNNNN	**L**	SELECT	0	T
01-01-02	3			0		N	NNNYN		SELECT	0	

ORDER BY Sort unavoidable

Prefetch Indicator