IIT School of Applied Technology

ILLINOIS INSTITUTE OF TECHNOLOGY

information technology & management

526 Data Warehousing

April 27, 2016

Week 14 Presentation

ITMD - 526

SQL Optimization for DW

Composite Index with EQUAL

```
SELECT  *
   FROM TAB1
WHERE COL1 = 'A'  AND COL2 = '112'
```

COL1	COL2	ROWID
A	110	10
A	111	11
A	112	5
A	113	18
A	114	54
A	115	23
A	116	29
A	117	25
A	118	26
A	119	30
A	120	19
A	121	32
B	110	41
B	111	45

INDEX

COL2	COL1	ROWID
110	A	10
110	B	41
110	C	57
110	D	81
111	A	11
111	B	39
111	C	76
111	D	98
112	A	5
112	B	73
112	C	89
112	D	77
113	A	18
113	B	65

INDEX

SQL Optimization for DW

Composite Index with EQUAL (cont'd)

```
SELECT  *
   FROM TAB1
WHERE COL1 = 'A'
    AND COL2 LIKE '111' AND '112'
```

COL1	COL2	ROWID
A	110	10
A	111	11
A	112	5
A	113	18
A	114	54
A	115	23
A	116	29
A	117	25
A	118	26
A	119	30
A	120	19
A	121	32
B	110	41
B	111	45

INDEX

COL2	COL1	ROWID
110	A	10
110	B	41
111	A	11
111	B	65
111	C	96
111	D	78
112	A	5
112	B	73
112	C	79
112	D	97
113	A	18
113	B	45
113	C	67
114	A	22

INDEX

ITMD - 526

SQL Optimization for DW

Composite Index: BETWEEN vs. IN

```
SELECT  * FROM TAB1
WHERE COL1 = 'A'
    AND COL2 between '111' AND '112'
```

COL2	COL1	ROWID
110	A	10
110	B	41
111	A	11
111	B	65
111	C	96
111	D	5
112	A	73
112	B	18
112	C	45
114	B	22

INDEX1

```
TABLE ACCESS BY ROWID TAB1
    INDEX RANGE SCAN INDEX1
```

```
SELECT  * FROM TAB1
WHERE COL1 = 'A'
    AND COL2 in ('111', '112')
```

COL2	COL1	ROWID
110	A	10
110	B	41
111	A	11
111	B	65
111	C	96
111	D	5
112	A	73
112	B	18
112	C	45
114	B	22

INDEX1

```
CONCATENATION
  TABLE ACCESS BY ROWID TAB1
    INDEX RANGE SCAN  INDEX1
  TABLE ACCESS BY ROWID TAB1
    INDEX RANGE SCAN INDEX1
```

SQL Optimization for DW

Composite Index: Adding a Column

```
SELECT *
  FROM TAB1
 WHERE A = '2' AND C = 51
```

A	B	ROWID
1	20	10
2	30	11
2	40	47
2	50	54
2	60	58
2	70	61
3	40	21

INDEX (A+B)

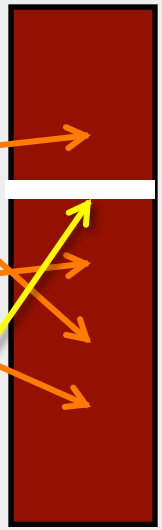


TABLE (TAB1)

A	B	C	ROWID
1	20	30	10
2	30	40	11
2	40	45	47
2	50	46	54
2	60	49	58
2	70	51	61
3	40	41	21

INDEX (A+B+C)

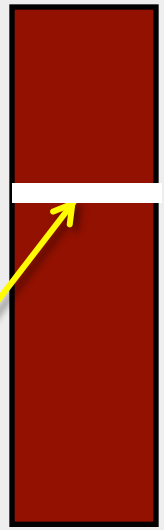
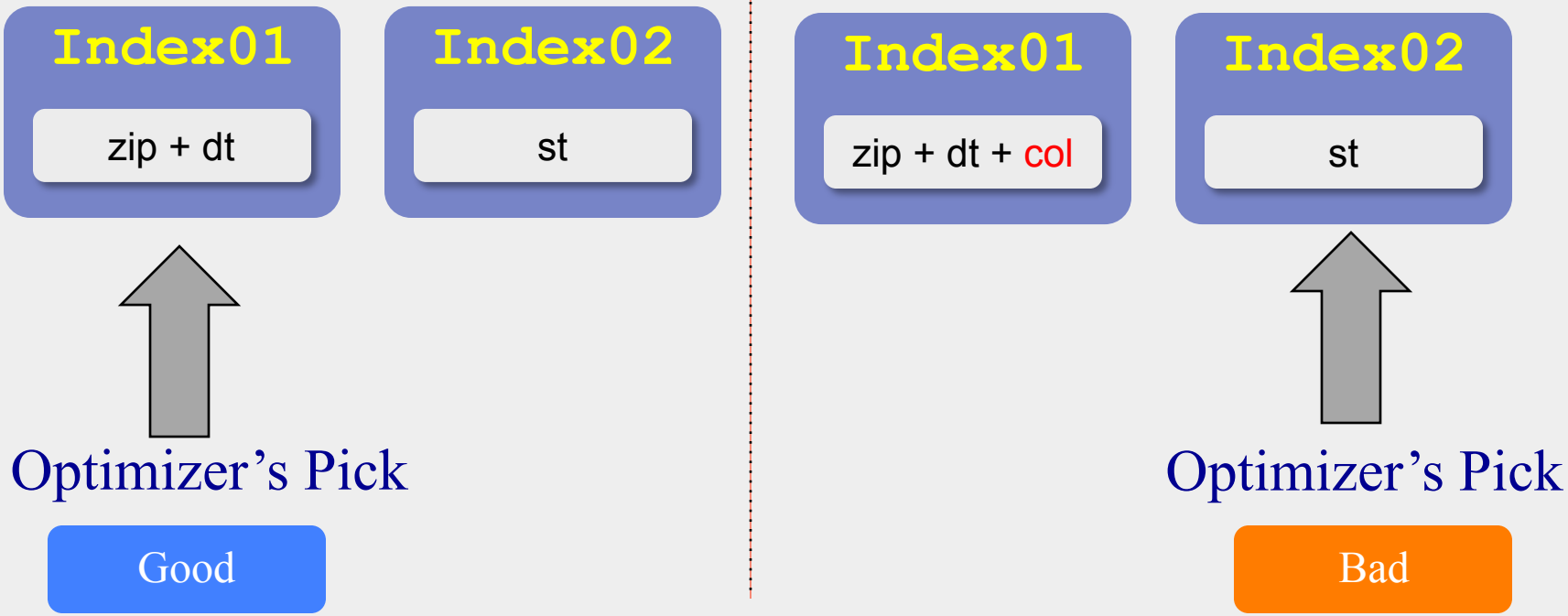


TABLE (TAB1)

SQL Optimization for DW

Composite Index: Adding a Column (cont'd)

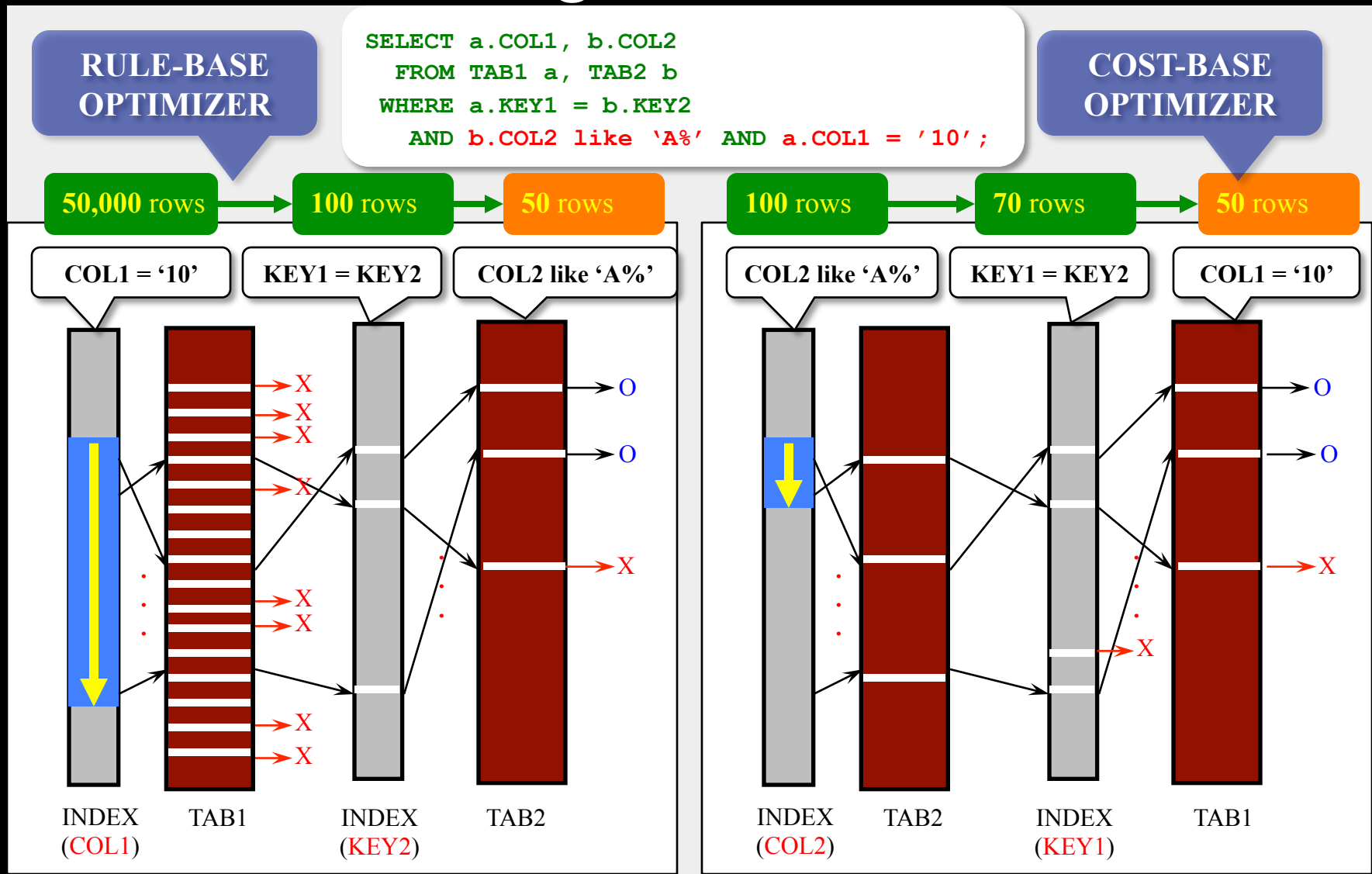
```
SELECT *
  FROM Sales
 WHERE zip = '60201'
   AND dt = '20141005'
   AND st = 'DELIVERED'
```



SQL Optimization for DW

JOIN – Driving Table

```
SELECT a.COL1, b.COL2
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND b.COL2 like 'A%' AND a.COL1 = '10';
```

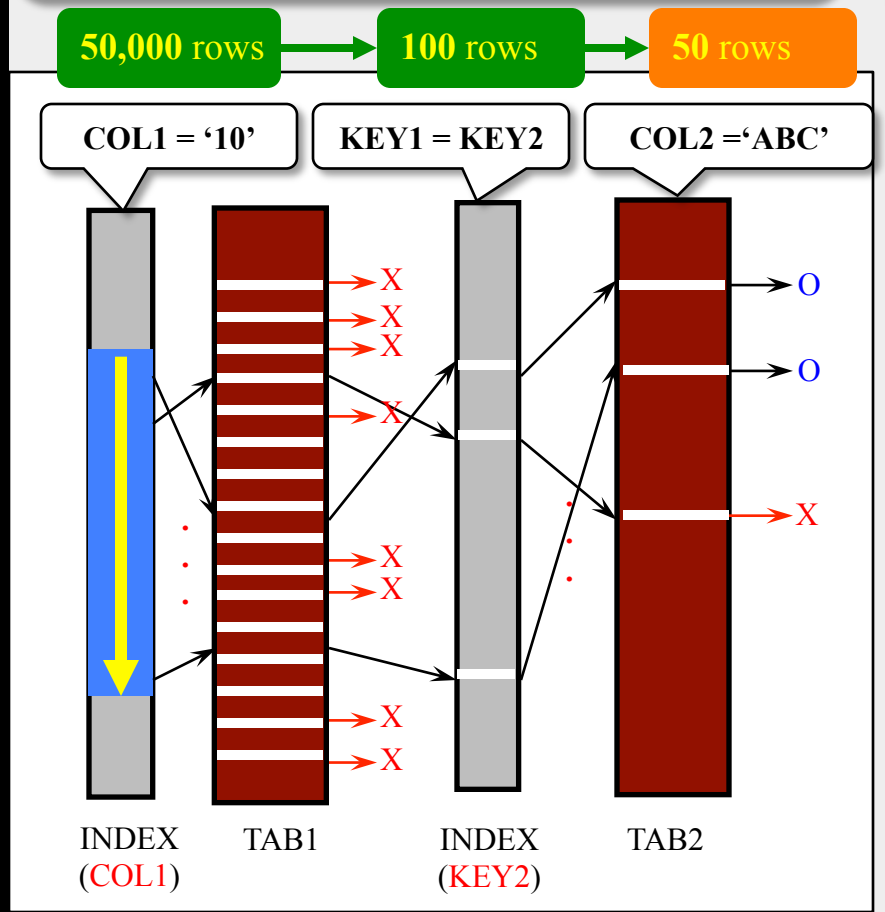


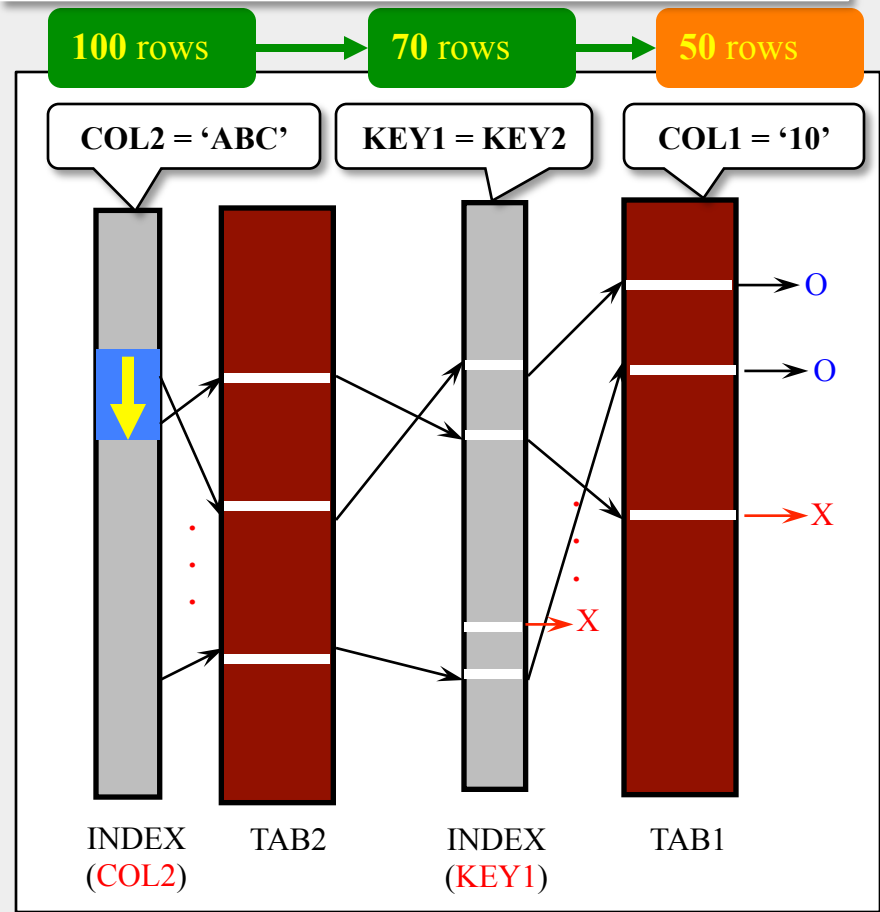
SQL Optimization for DW

JOIN – Driving Table (cont'd)

```
SELECT a.COL1, b.COL2
  FROM TAB2 b, TAB1 a
 WHERE a.KEY1 = b.KEY2
   AND b.COL2 = 'ABC' AND a.COL1 = '10';
```

TUNING OPTIONS
- ANALYZE table (RULE-BASE Optimizer)
- HINT (/*+ ORDERED */)
- SUPPRESSING (a.COL1||'' = '10')





SQL Optimization for DW

JOIN: Driving Table Example

```
SELECT INVNO, INVDATE, CUSTNAME
  FROM CUSTOMER Y, INVOICE X
 WHERE X.CUSTNO = Y.CUSTNO
   AND X.INVDATE = '20151013'
   AND Y.NATION  = 'USA'
```

1 rows, 0.14 sec

```
NESTED LOOPS
  TABLE ACCESS BY ROWID CUSTOMER
    INDEX RANGE SCAN IX_NATION
  TABLE ACCESS BY ROWID INVOICE
    AND-EQUAL
      INDEX RANGE SCAN IX_CUSTNO
      INDEX RANGE SCAN IX_INVDATE
```

```
SELECT INVNO, INVDATE, CUSTNAME
  FROM INVOICE X, CUSTOMER Y
 WHERE X.CUSTNO = Y.CUSTNO
   AND X.INVDATE = '20151013'
   AND Y.NATION  = 'USA'
```

1 rows, 0.03 sec

```
NESTED LOOPS
  TABLE ACCESS BY ROWID INVOICE
    INDEX RANGE SCAN IX_INVDATE
  TABLE ACCESS BY ROWID CUSTOMER
    INDEX UNIQUE SCAN IX_CUSTNO
```

SQL Optimization for DW

JOIN: Use of Index

```
SELECT INVNO, INVDATE, CUSTNAME
  FROM CUSTOMER Y, INVOICE X
 WHERE X.CUSTNO = Y.CUSTNO
   AND X.INVDATE = '20151013'
   AND Y.NATION  = 'USA'
```

1 rows,
0.14 sec

```
NESTED LOOPS
  TABLE ACCESS BY ROWID CUSTOMER
    INDEX RANGE SCAN IX_NATION
  TABLE ACCESS BY ROWID INVOICE
    AND-EQUAL
      INDEX RANGE SCAN IX_CUSTNO
      INDEX RANGE SCAN IX_INVDATE
```

```
SELECT INVNO, INVDATE, CUSTNAME
  FROM CUSTOMER Y, INVOICE X
 WHERE RTRIM(X.CUSTNO) = Y.CUSTNO
   AND X.INVDATE = '20151013'
   AND Y.NATION  = 'USA'
```

1 rows,
0.14 sec

```
NESTED LOOPS
  TABLE ACCESS BY ROWID INVOICE
    INDEX RANGE SCAN IX_INVDATE
  TABLE ACCESS BY ROWID CUSTOMER
    INDEX UNIQUE SCAN PK_CUSTNO
```

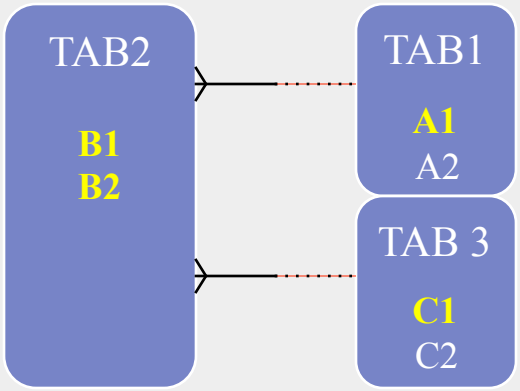
```
SELECT INVNO, INVDATE, CUSTNAME
  FROM CUSTOMER Y, INVOICE X
 WHERE RTRIM(X.CUSTNO) =
       RTRIM(Y.CUSTNO)
   AND X.INVDATE = '20151013'
   AND Y.NATION  = 'USA'
```

rows,
sec

?

SQL Optimization for DW

JOIN: Deciding Access Path




```
SELECT A1,A2,.., B1,B2,..,C1,C2,..
  FROM TAB1 t1, TAB2 t2, TAB3 t3
 WHERE t1.A1 = t2.B1
   AND t3.C1 = t2.B2
   AND t1.A2 = '10'
   AND t2.B2 LIKE 'WB%'
```

Driving	Access Path	Index Strategy
TAB1	A2 = '10' B1 = A1 and B2 like 'WB%' C1 = B2	• A2 • B1 + B2 • C1
TAB2	B2 like 'WB%' C1 = B2 A1 = B1 and A2 = '10'	• B2 • C1 • A1 + A2
TAB3	Full table scan B2 = C1 and B2 like 'WB%' A1 = B1 and A2 = '10'	• B2 • A1 + A2

SQL Optimization for DW

JOIN: Deciding Access Path (cont'd)

```
SELECT A1,A2,.., B1,B2,..,C1,C2,..
  FROM TAB1 t1, TAB2 t2, TAB3 t3
 WHERE t1.A1 = t2.B1
   AND t3.C1 = t2.B2
   AND t1.A2 = '10'
   AND t2.B2 LIKE 'WB%'
```

```
SELECT A1,A2,.., B1,B2,..,C1,C2,..
  FROM TAB1 t1, TAB2 t2, TAB3 t3
 WHERE t1.A1 = t2.B1
   AND t3.C1 = t2.B2
   AND t1.A2 = '10'
   AND t3.C1 LIKE 'WB%'
```

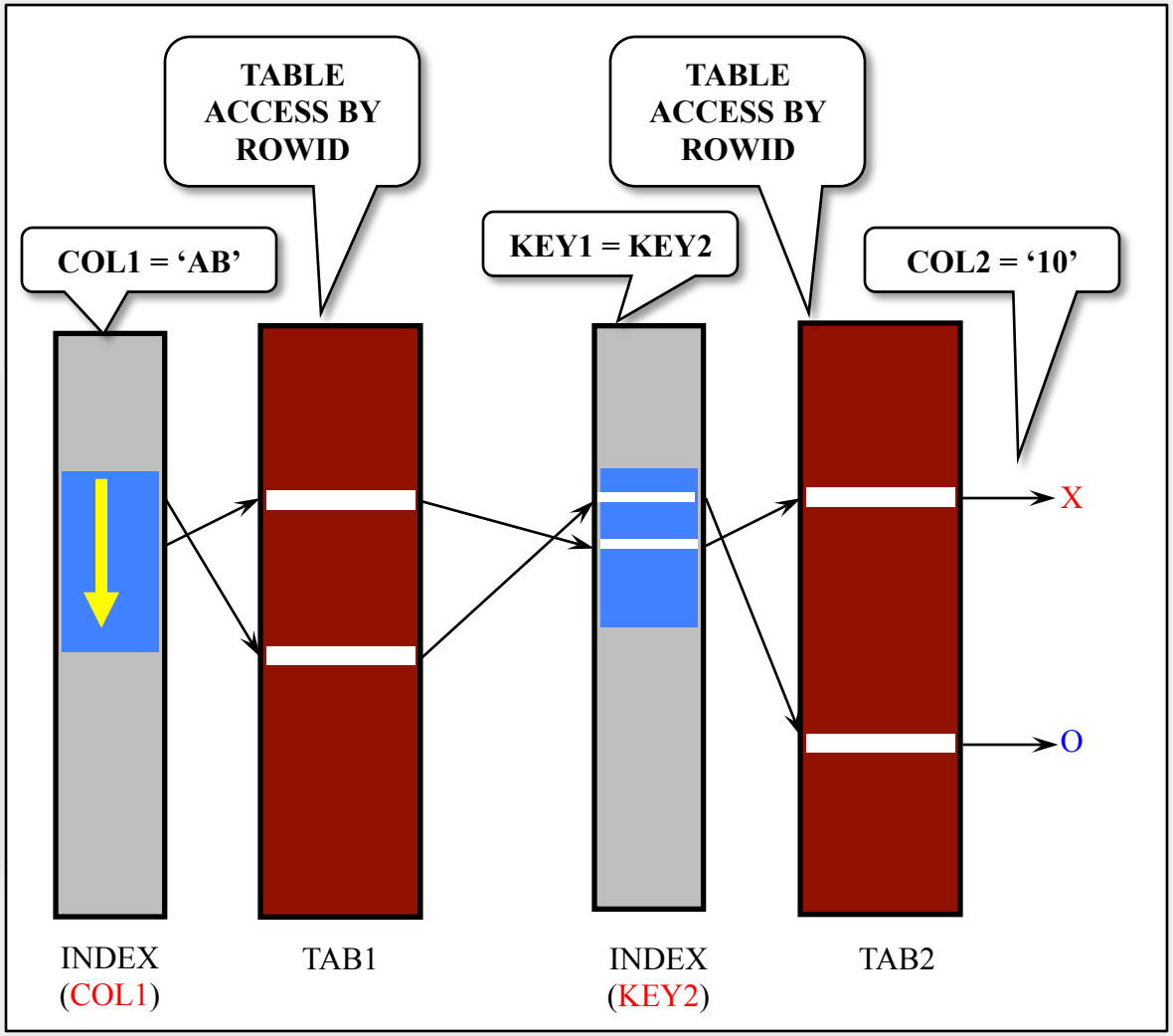
Driving	Access Path	Index Strategy
TAB3	Full table scan B2 = C1 and B2 like 'WB%' A1 = B1 and A2 = '10'	• B2 • A1 + A2
TAB3	C1 like 'WB%' C1 = B2 A1 = B1 and A2 = '10'	• C1 • B2 • A1 + A2

SQL Optimization for DW

JOIN – Nested Loop JOIN

```
SELECT a.COL1, b.COL2
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

- Sequential
- Partial Range Scan
- Dependency
- Random Table I/O
- Fit for small range
- Indexing on join is important
- OLTP

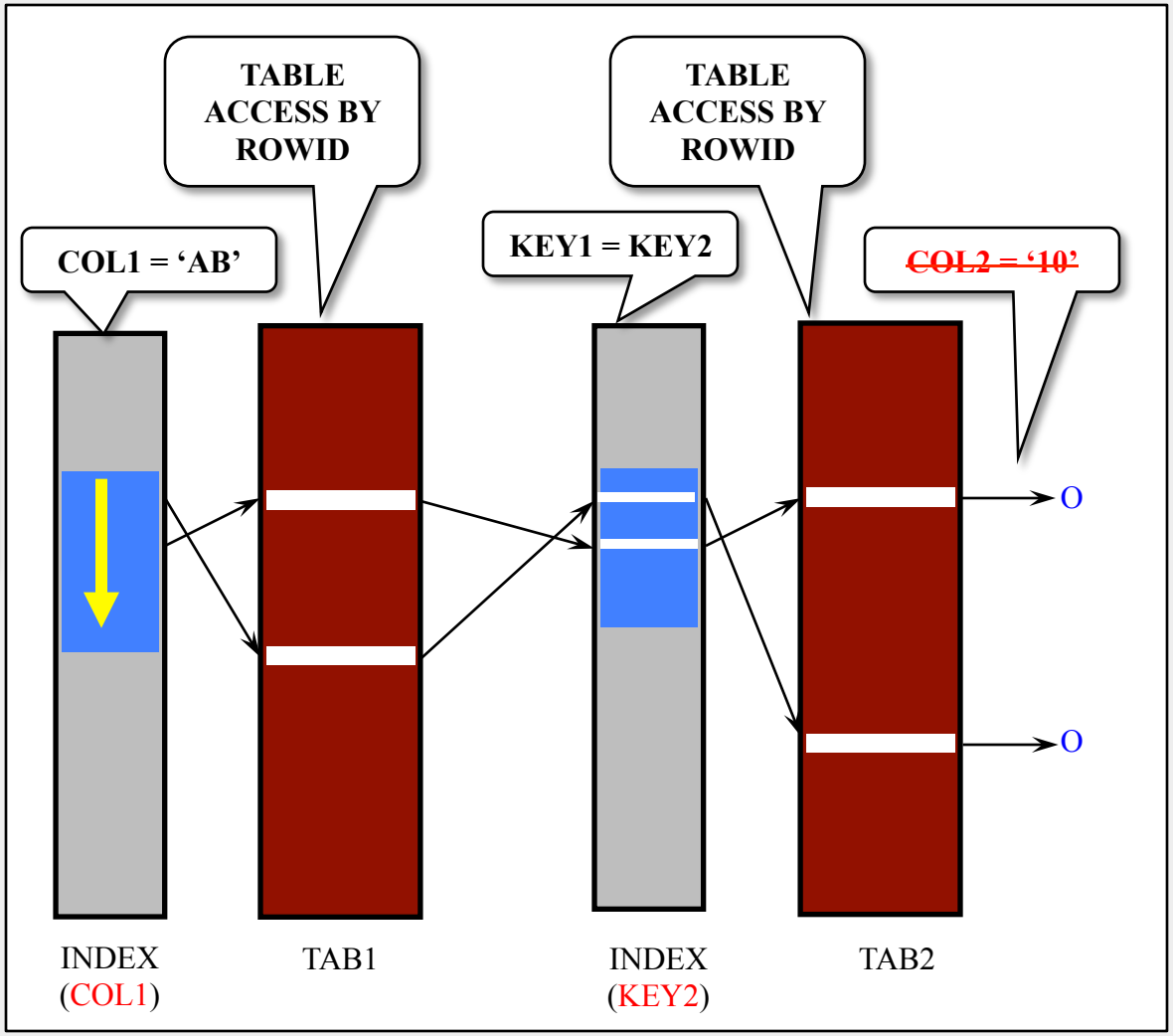


SQL Optimization for DW

JOIN – Nested Loop JOIN (cont'd)

```
SELECT a.COL1, b.COL2
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

Eliminating b.col2='10' condition has positive impact to the response time

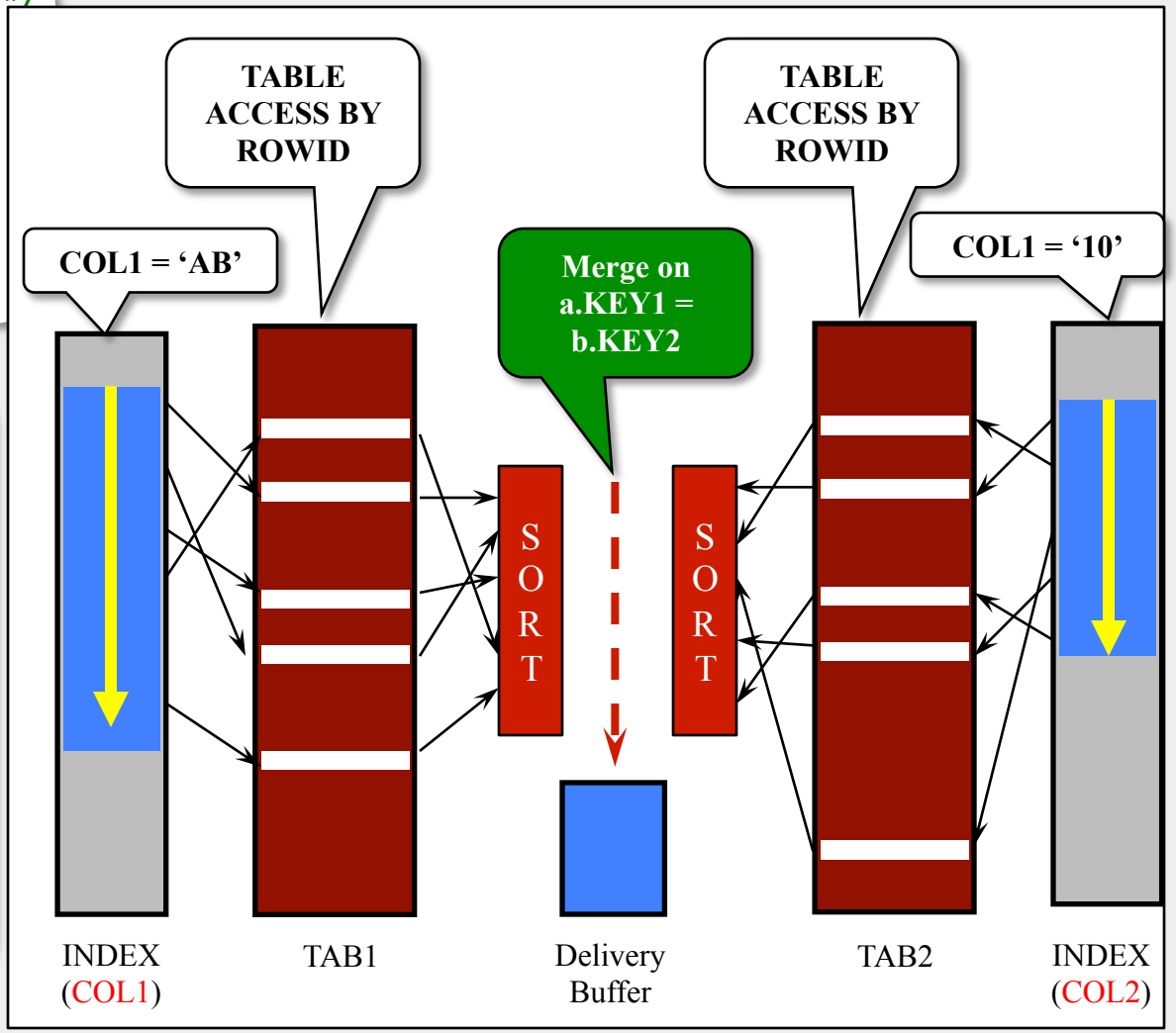


SQL Optimization for DW

JOIN – Sort Merge JOIN

```
SELECT /*+ use_merge(a b) */
       a.COL1, b.COL2, …
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

- Parallel
- Full Range Scan
- Constrain on both tables needed
- Scan access on sort
- Indexing on join is not important
- Fit for wide range
- DW

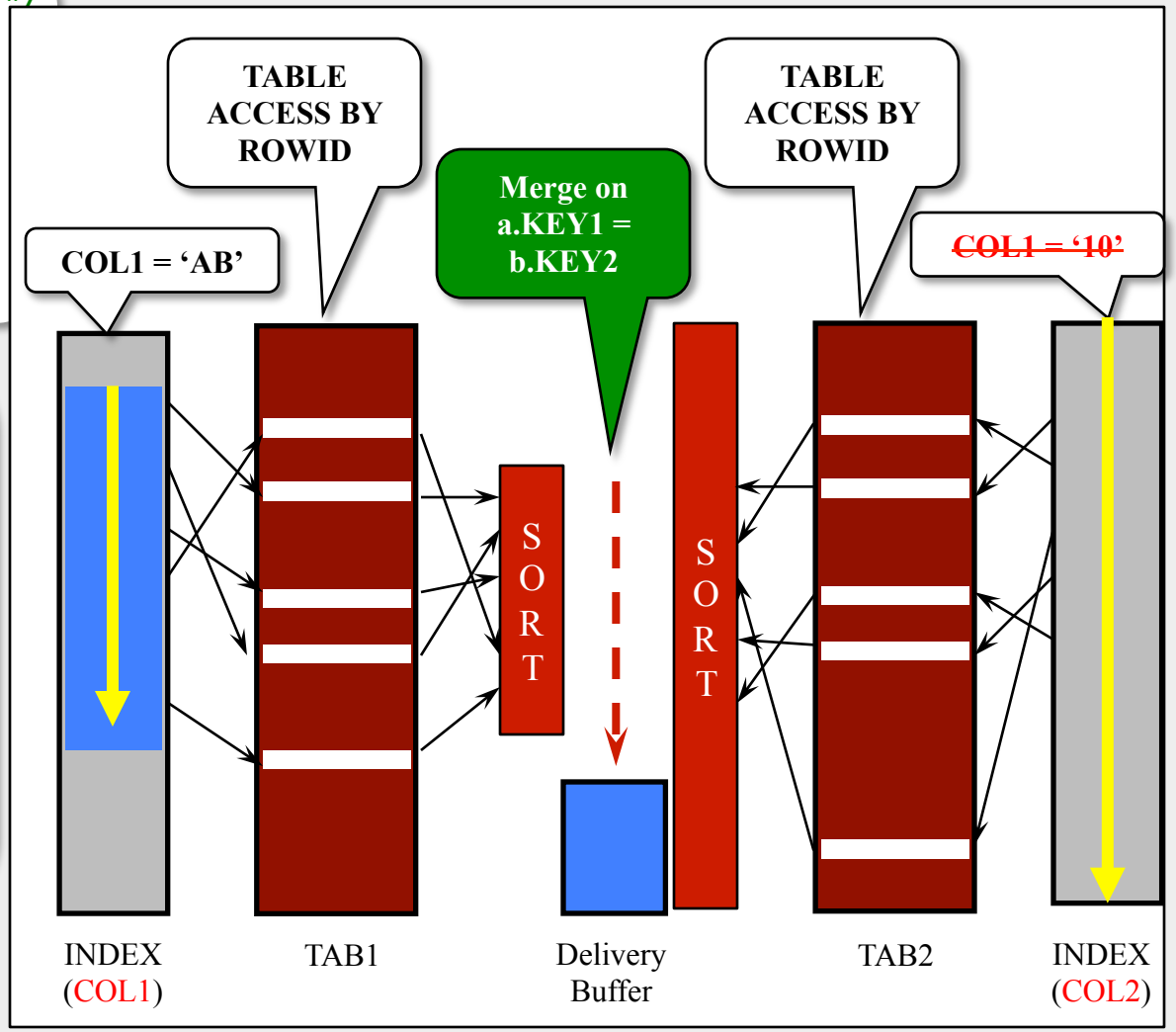


SQL Optimization for DW

JOIN – Sort Merge JOIN (cont'd)

```
SELECT /*+ use_merge(a b) */
       a.COL1, b.COL2, …
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

Sort merge becomes ineffective when one of the sort buffer size is much bigger than the other
→ Hash merge join can be an option



SQL Optimization for DW

JOIN – Execution Plan Examples

- Throughput

```
SELECT a.COL1, b.COL2
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

```
MERGE JOIN (SORT)
  TABLE ACCESS BY ROWID TAB1
    INDEX RANGE SCAN IX_COL1
  TABLE ACCESS BY ROWID TAB2
    INDEX RANGE SCAN IX_COL2
```

- Response time

```
SELECT --+ RULE
       a.COL1, b.COL2
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
   AND a.COL1 = 'AB'
   AND b.COL2 = '10';
```

```
NESTED LOOPS
  TABLE ACCESS BY ROWID TAB1
    INDEX RANGE SCAN IX_COL1
  TABLE ACCESS BY ROWID TAB2
    INDEX UNIQUE SCAN IX_KEY2
```

SQL Optimization for DW (cont'd)

JOIN – Execution Plan Examples

- Throughput

```
SELECT b.COL3, SUM(…)
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
 GROUP BY b.COL3;
```

```
SORT GROUP BY
  MERGE JOIN
    SORT JOIN
      TABLE ACCESS FULL TAB1
    SORT JOIN
      TABLE ACCESS FULL TAB2
```

- Response time

```
SELECT --+ RULE
       b.COL3, SUM(…)
  FROM TAB1 a, TAB2 b
 WHERE a.KEY1 = b.KEY2
 GROUP BY b.COL3;
```

```
SORT GROUP BY
  NESTED LOOPS
    TABLE ACCESS FULL TAB1
    TABLE ACCESS BY ROWID TAB2
      INDEX RANGE SCAN KEY2
```