Index Design in a Busy System

Larry Kintisch & Tapio Lahdenmäki

BWDB2UG 9/13/2006

Agenda [70 + 70 minutes]:

- A "simple" query: a difficult index
- Introduction to Statistics of simple queues
- Effect of queueing on a DASD system
- Query with random "touches"
- Importance of Index improvement
--break
- JOIN indexes and JOIN sequence
- Star Join Issues & Summary tables
- Statistics for Index inserts
- Best freespace policy for Indexes
- Reorg Index policy for busy systems
- A last word: Lock Waits in a Busy System

I.2.

Quick Upper-Bound Estimate (QUBE)

Redbook SG24-2549 published August 1995

- provides simple method to estimate LRT: raise a flag
- easy to compare alternatives; simplifications, assumptions
- updated to reflect faster systems; basic concept is sound
- TS is a "touch" of next seql. row; TR is random row access

ET = TR x 10 ms + TS x 0.01 ms + F x 0.1 ms

CPU = TR x 50 mics + TS x 5 mics + F x 50 mics

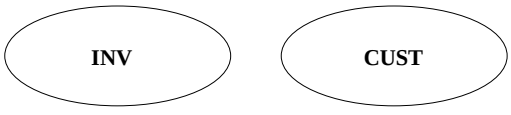
ET = Elapsed time (SQL)
TR = Number of random touches
F = Number of FETCH calls

CPU = CPU time (SQL)
TS = Number of sequential touches

1.19 I.3

A Simple Query

X1: P X2: C

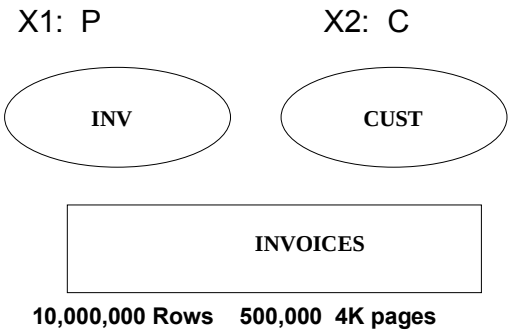


10,000,000 Rows 500,000 4K pages

```
SELECT CUST, INV#,  IDATE, ITOTL, STORE, CLERK, ZIP
FROM  INVOICES
WHERE  ZIP = :hv-zip
  AND  IDATE BETWEEN :hv-first AND :hv-last
  AND  ITOTL >= :inv-totl
ORDER BY CUST, IDATE desc
OPTIMIZE FOR 20 ROWS
```

I.4

Do we need an Improved Index for this Query?



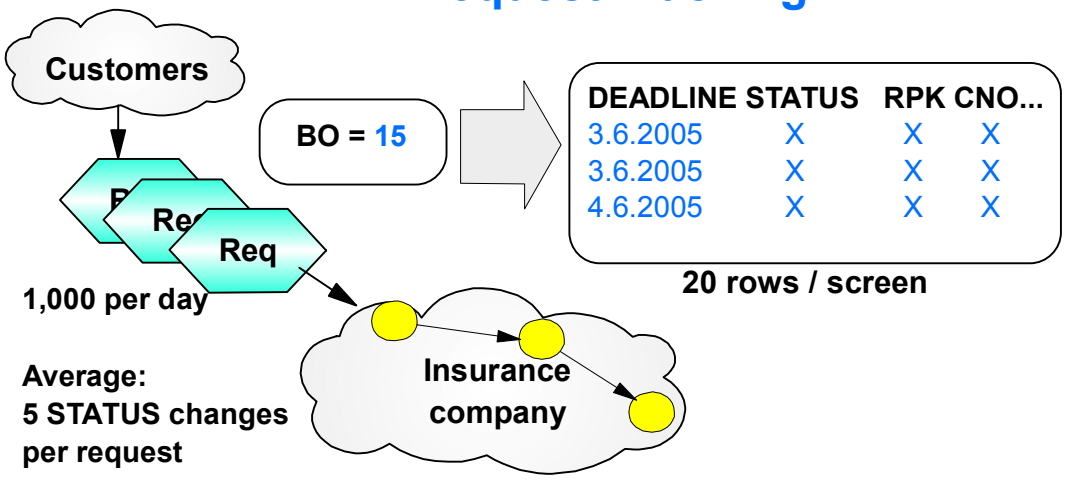
Use an existing index?
Add to an existing index?
Do a Table scan and sort?
A New index that takes
advantage of the cluster sequence?
A New index that has all of the
desired key values close together?
A New index that avoids
sorting the results?

How would you choose?

```
SELECT CUST, INV#,  IDATE, ITOTL, STORE, CLERK, ZIP
FROM  INVOICES
WHERE  ZIP = :hv-zip
  AND  IDATE BETWEEN :hv-first AND :hv-last
  AND  ITOTL >= :inv-totl
ORDER BY CUST, IDATE desc
OPTIMIZE FOR 20 ROWS
```

I.5

Inadequate Indexing: Case Study
Request Tracking

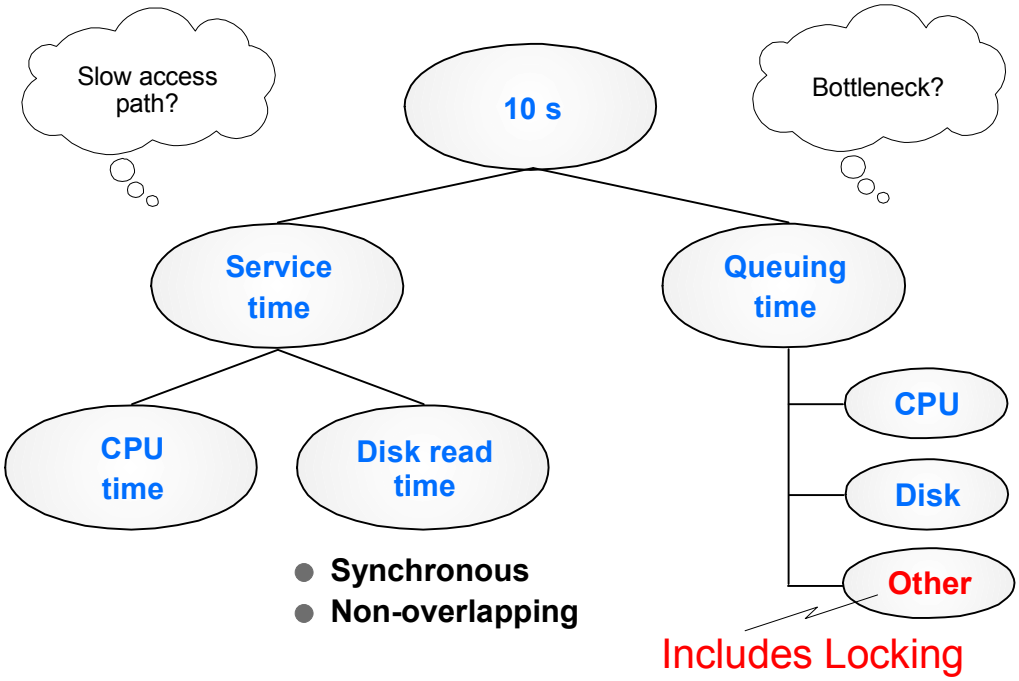


Primary key of REQUEST = RPK, foreign keys CNO and BO
BO = Branch office (100 branch offices, the largest one covers 10% of requests)
STATUS: 1...9 (9 = Closed)
DL = Deadline

DL = 31.12.2099
BO = Latest BO
99%

1.04 C.6

Components of Response Time



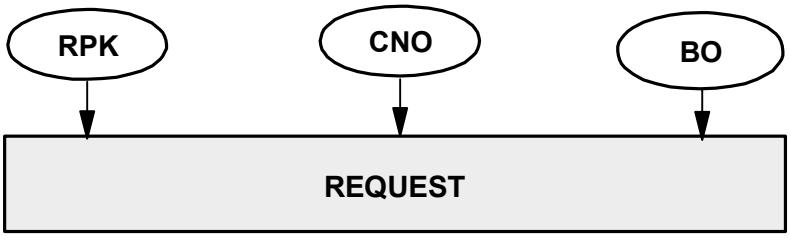
1.05 C.7

Common Transaction

```
SELECT   DL, STATUS, RPK, CNO, C1, C2
FROM     REQUEST
WHERE    STATUS < 9
AND
         BO = :BO
ORDER BY DL
```

FF = 1 %

FF = 0...10 %

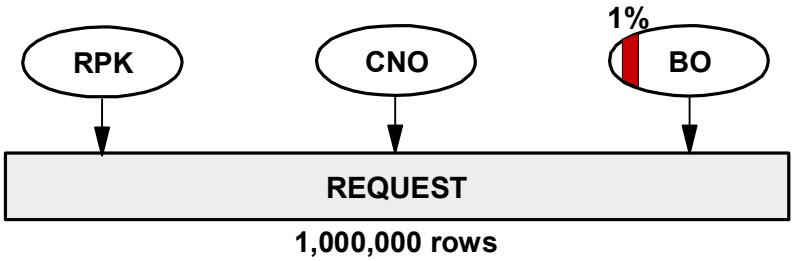


1,000,000 rows, average length 300 bytes

FF = Filter Factor: 0...100%

1.06 C.8

Which One Faster?



Alternative 1

Read 1,000,000 table rows

Pick rows that satisfy both predicates

Sort 100 result rows

Alternative 2

Read index slice

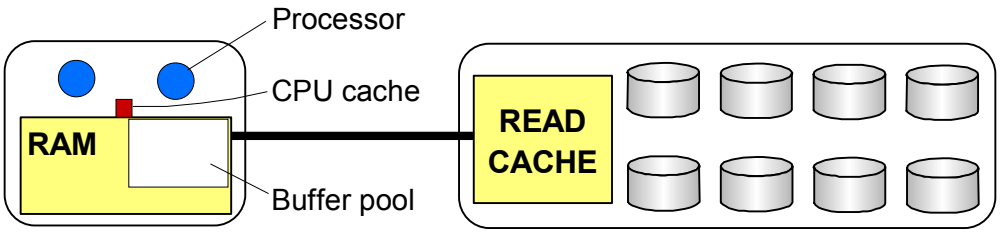
Read 10,000 table rows

Pick rows that satisfy STATUS < 9

Sort 100 result rows

1.07 C.9

Full Table Scan in 2005



I/O time

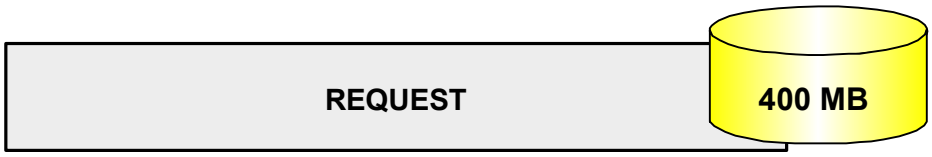
- DBMS and the disk subsystem read ahead -- lots of pages with one rotation
- Not all pages at once -- just trying to stay ahead: when the program needs a page it should be in the buffer pool
- If sequential read speed **40 MB/s**, I/O time per 4K page 0.1 ms; if 10 rows per page, I/O time per row 0.01 ms = **10 mics [micro-secs]**

CPU time

- Rule of thumb: CPU time per examined row = **5 mics** with sequential read
- FETCH (move qualifying row to application pgm) may take **50 mics** of CPU time

1.08 C.10

Alternative 1



1,000,000 rows

CPU time

- Rule of thumb: 1,000,000 x 5 mics = **5 s** just to examine 1,000,000 rows
- CPU load point of view: Transactions that require more than **0.1 s** of CPU time should be rare

I/O time

- 1,000,000 x 300 bytes + free space may be 400 MB
- If sequential read speed 40 MB/s, full table scan takes **10 s**
  400 MB / 40 MB/s = 10 s

Conclusion

- Too expensive & too slow

1.09 C.11

Introducing Statistics for Queues

Suppose an escalator can accept one person per second, [its service time S = 1.0 sec]:

If one person steps onto the escalator every 10 seconds then the device is busy 10% of the time [its utilization factor u = .10]
If one person steps onto the escalator every 5 seconds, u=.20
If one person steps onto the escalator every 2 seconds, u=.50
If, like clockwork, a person appears and steps onto the escalator every second, u=1.0, fully utilized.

C.12.

Queues will Form...

When fully utilized, clockwork arrivals greater than one per second will form a queue, that will get longer over time until arrivals drop below 1/sec.

If instead, a person appears *randomly* on average once per second then:
- in some seconds 1 person arrives
- in some seconds 0 persons arrive
- in some seconds 2 persons arrive: queue forms
- in some seconds 3 persons arrive: queue forms

Random arrivals with an average inter-arrival time are said to follow a Poisson distribution.

C.13

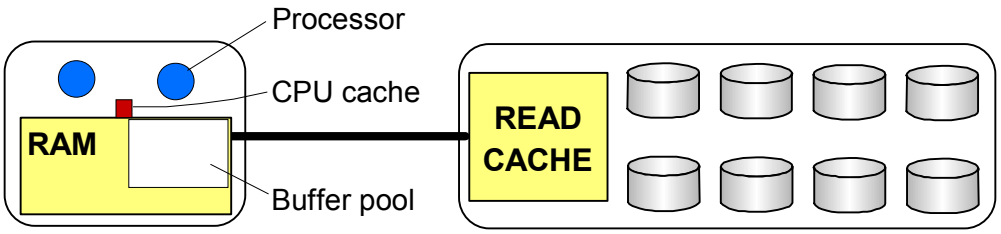
How long will a request for service have to wait?

Q = average queuing time = $(u/(1-u)) \times S$
where u = the fraction busy for the server
and S = the average service time

For our escalator with S = $\underline{1}$, if random arrivals come 5 every 10 seconds on average, u = .5:
Q = $(.5/(1-.5)) \times \underline{1} = 1$; a person would expect to wait 1 second!
If random arrivals come 6 every 10 seconds, u=.6
Q = $(.6/(1-.6)) \times \underline{1} = 1.5$ seconds waiting
If random arrivals come 9 every 10 seconds, u=.9
Q = $(.9/(1-.9)) \times \underline{1} = 9$ seconds waiting

C.14

Random Read in 2005



Disk I/O time

ms: 3 Q + 4 Sk + 2 rot +1 trsf

- If needed page not in pool: disk read
- If needed page in read cache: I/O time may be **1 ms**
- Random read from disk drive may take **10 ms**

Serious

CPU time

- Retrieving a row and evaluating it may take **50 mics of** CPU time (random read)
- FETCH one row may take **50 mics** of CPU time -- as with sequential read

1.10 C.15

Random Read

Queuing (Q)	3 ms
Seek	4 ms
Half a rotation	2 ms
Transfer	1 ms
Total I/O time	10 ms

Depends on drive busy

S = Service time

One random read keeps a drive busy for 6 ms

Q = (u / (1-u)) x S

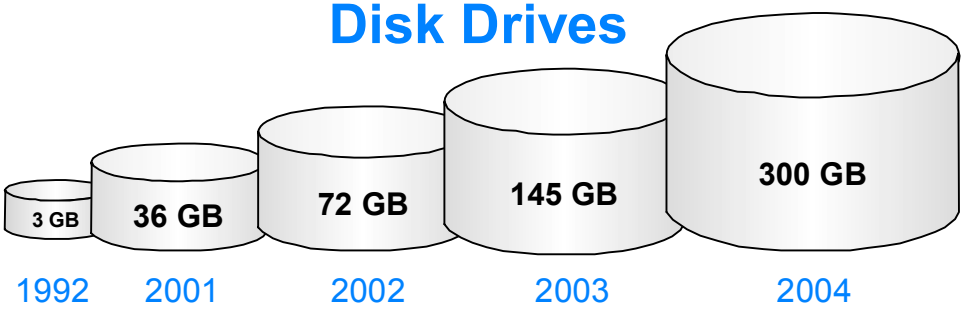
Q = Average queuing time
u = Average drive busy
S = Average service time

50 random reads a second

u = 50 read/s x 0.006 s/read
= 0.3
Q = (0.3 /(1- 0.3)) x 6 ms
= 3 ms

1.11 C.16

Disk Drives



- **Storage density grows dramatically**
- **Sequential I/O getting faster**
- **Random I/O remains slow (and may even become slower)**

Q = (u / (1-u)) x S

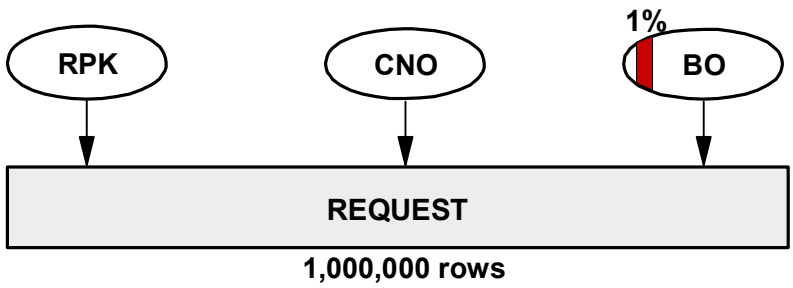
Q = Average queuing time
u = Average drive busy
S = Average service time

u 30% to 60%
Q 3 ms to 9 ms
Random read 10 ms to 16 ms

If 80% busy, Q = 16, SR=21ms ; if 92% busy, Q= 69, SR=74ms; 95%->SR= 119ms

1.12 C.17

Alternative 2

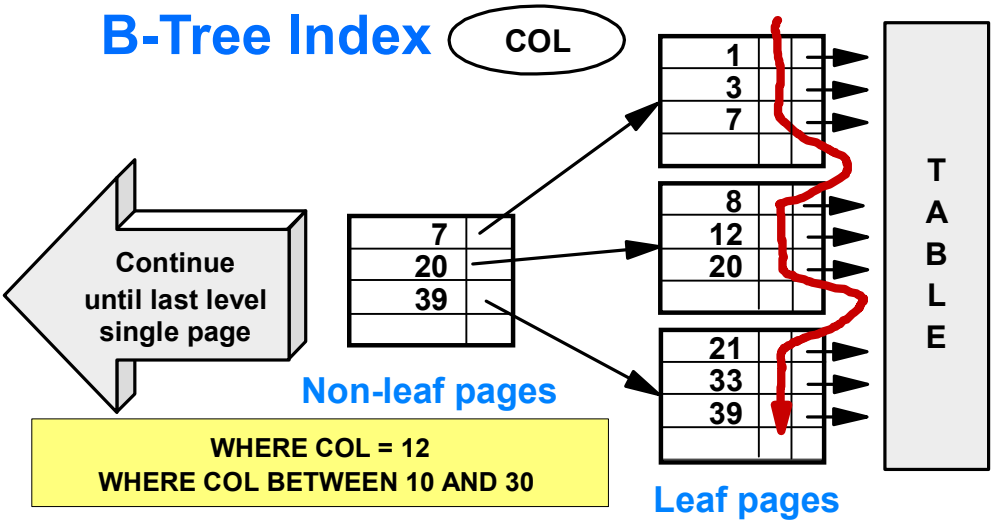


- Average branch office has **10,000 table rows** (FF = 1%)
- If these rows not adjacent, up to 10,000 random reads from disk
- I/O time up to **10,000 x 10 ms = 100 s**
- Alternative 2 acceptable only if table rows in BO sequence
  Read table **slice**, fast sequential read

How does DBMS find table rows with BO = :BO?

1.13 C.18

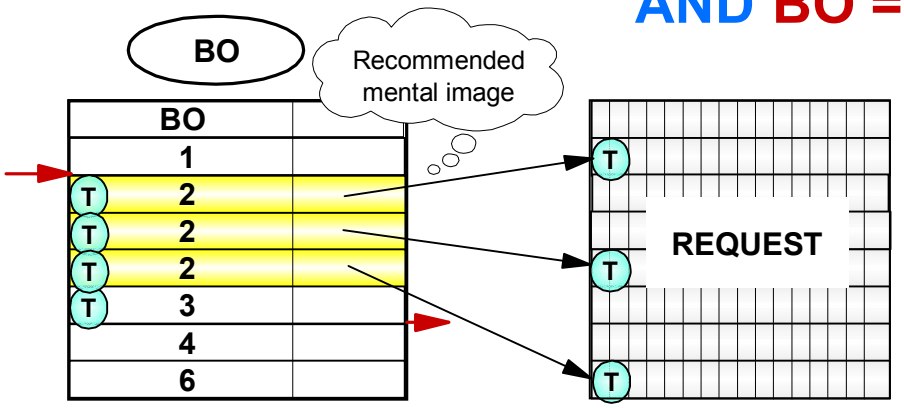
B-Tree Index



- Normally 3 levels if 1,000,000 table rows
- Number of non-leaf pages much lower than number of leaf pages
- Non-leaf pages tend to stay in pool with current hardware
- Reasonable (2005): Ignore cost of non-leaf page processing

1.14. C.19

Touches: WHERE STATUS < 9 AND BO = 2



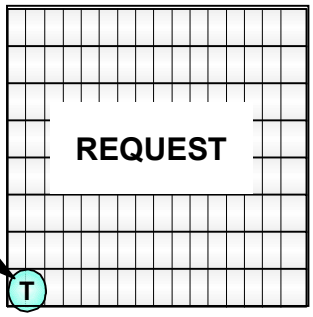
1.15. C.20

* For each query ask the "Basic Question"

WHERE STATUS < 9 AND BO = 2

Better index

	STATUS	BO	
T	1	3	
T	1	88	
T	3	2	
T	4	70	
T	9	1	
	9	2	
	9	2	



M column — S column

STATUS < 9 defines index slice	BO = :BO evaluated in index
Matching	Screening

Basic Question:
All predicate columns in one index (created or planned)?

1.18. C.21

Quick Upper-Bound Estimate (QUBE)

ET = TR x 10 ms + TS x 0.01 ms + F x 0.1 ms

CPU = TR x 50 mics + TS x 5 mics + F x 50 mics

or factored....

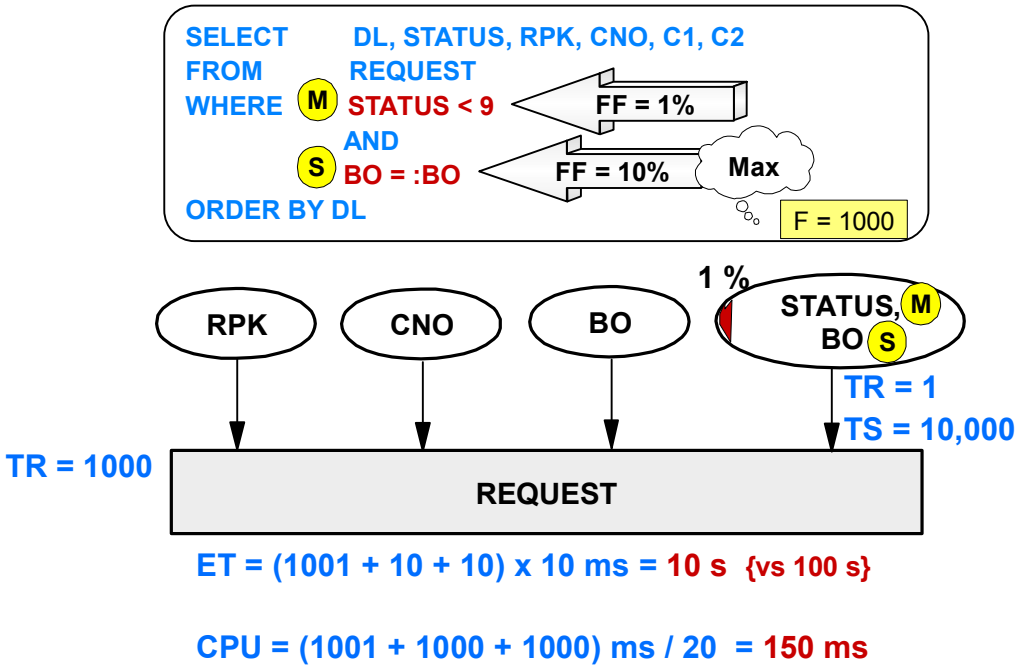
ET = (TR + TS/1000 + F/100) x 10 ms

CPU = (TR + TS/10 + F) ms / 20

ET = Elapsed time (SQL) — CPU = CPU time (SQL)
TR = Number of random touches — TS = Number of sequential touches
F = Number of FETCH calls

1.19 Q.22

Semi-Fat Index (STATUS, BO)

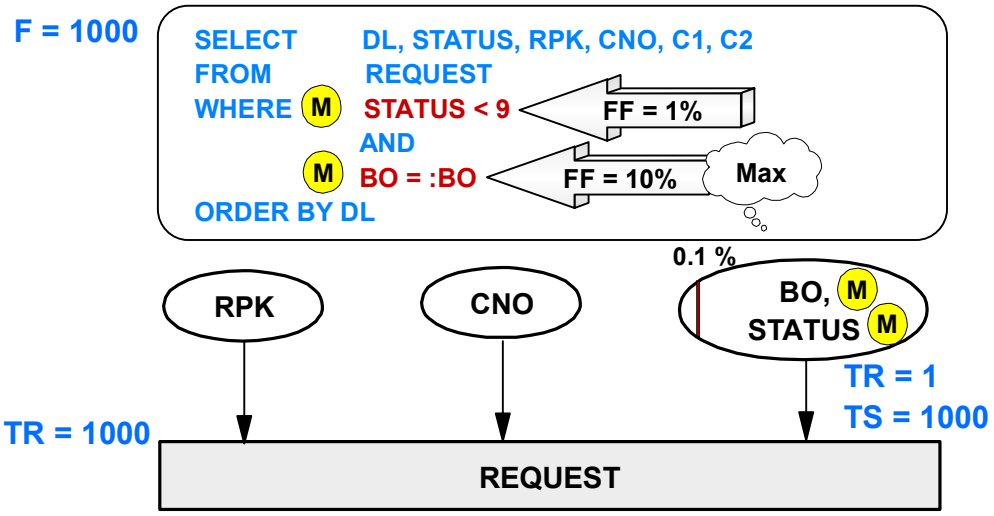


ET = (1001 + 10 + 10) x 10 ms = 10 s {vs 100 s}

CPU = (1001 + 1000 + 1000) ms / 20 = 150 ms

1.21 Q.23

Semi-Fat Index (BO, STATUS)



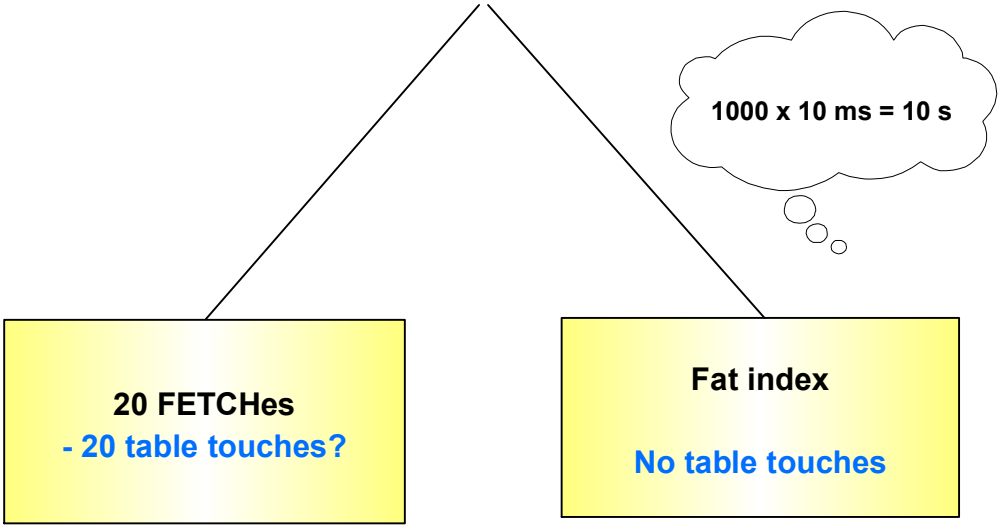
ET = (TR + TS/1000 + F/100) x 10 ms = 10 s

CPU = (TR + TS/10 + F) ms / 20 = 101ms {vs 150}

1.23 Q.24

10 s Too Long - What Next?

The problem: 1000 random table touches



1.24 Q.25

20 FETCHes - 20 Table Touches?

BO = AND STATUS <	SORT	MC	SC
(DL, BO, STATUS)	N	0	2
(BO, DL, STATUS)	N	1	1
(BO, STATUS, DL)	Y	2	0

SORT YES/NO

MC = Number of matching columns

Must have an index with result rows in correct order

ORDER BY DL

Must ask optimizer to make cost estimates for 20 FETCHes

Application program may have to reposition

Next 20 result rows

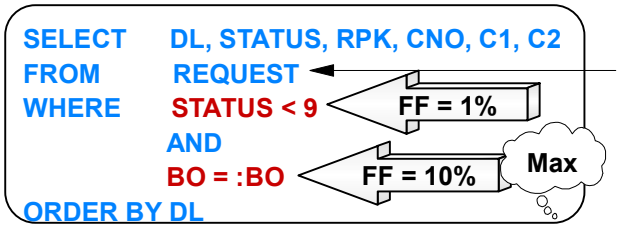
Should have unique index

Repositioning performance

..and then:
Filter Factor Pitfall

1.26 Q.26

Fat Index (BO, STATUS, RPK, DL, CNO, C1, C2)



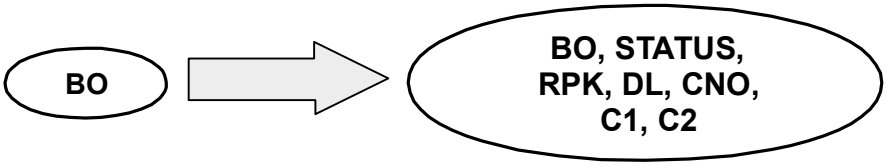
MC = 2 SC = 0 IXONLY = Y SORT = Y	Index		Table	
	TR	TS	TR	TS
	1	1000	-	-

	TR		TS/1000		F/100			
ET = (	1	+	1	+	10	) x 10 ms =	.12 s	{vs 10s}

	TR		TS/10		F			
CPU = (	1	+	100	+	1000	) ms / 20 =	55 ms	{vs 101ms}

1.27 Q.27

Too Expensive?



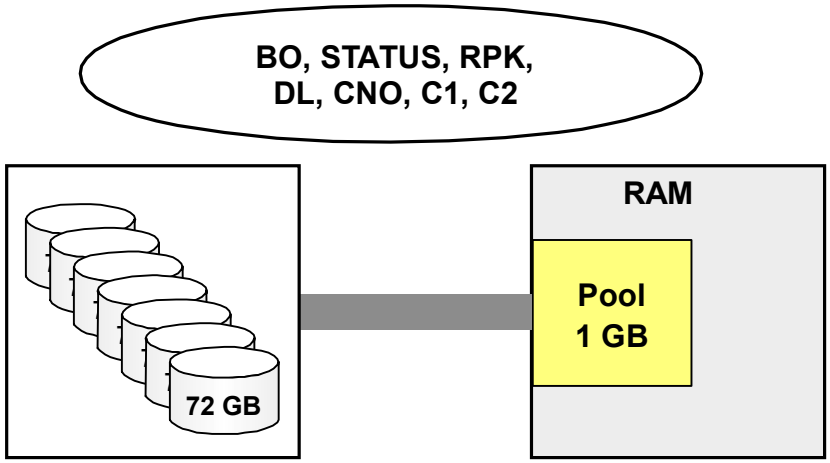
1,000,000 rows
80 bytes per row

- Disk space
- RAM (Non-leaf pages)
- INSERT
- UPDATE
- DELETE
- Index reorg (rebuild)
- Something else?

1000 new rows per day
5000 STATUS updates per day

1.28 Q.28

Disk Space and RAM



1,000,000 rows x 80 bytes / row
Assume free space 50%
Disk space 0.2 GB
Assume 50 e / GB / month
0.2 GB = 10 e / month

Assume non-leaf pages
1 % of leaf pages
0.01 x 200 MB = 2 MB
? e / MB / month

Add Index Row

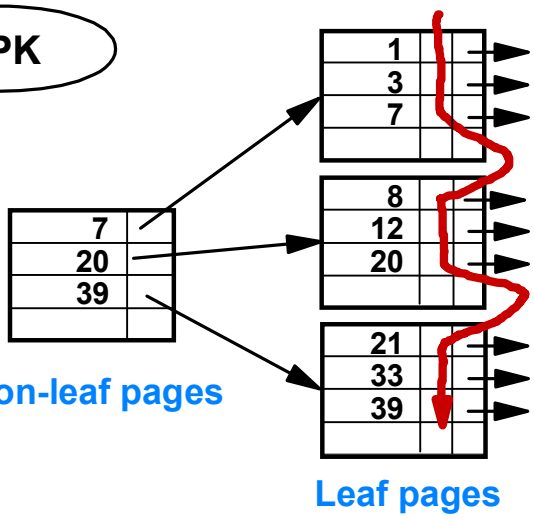
1 **Add RPK 10**

RPK

Non-leaf pages in pool
Read leaf page from disk
QUBE: TR = 1 (ET = 10 ms)

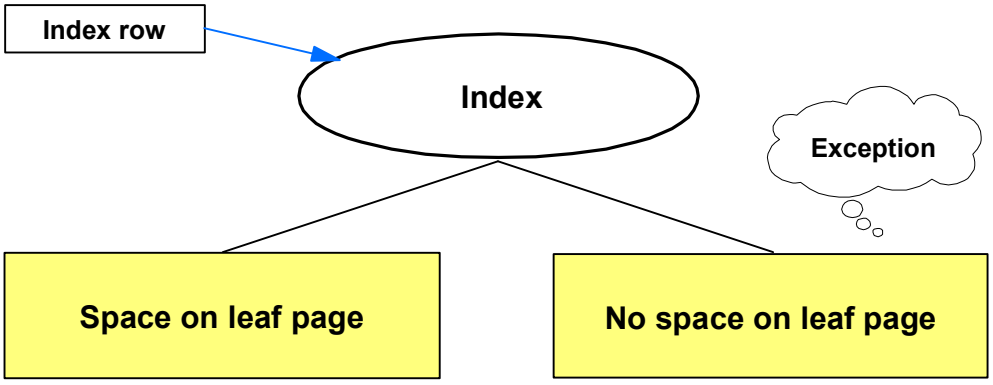
Later...

Write leaf page to disk
Does not affect response time
Does increase drive load



What if many index columns?
INSERT slows down?

Add Row To Fat Index



INSERT response time up by about 10 ms (1 TR)

One random I/O moves the whole leaf page

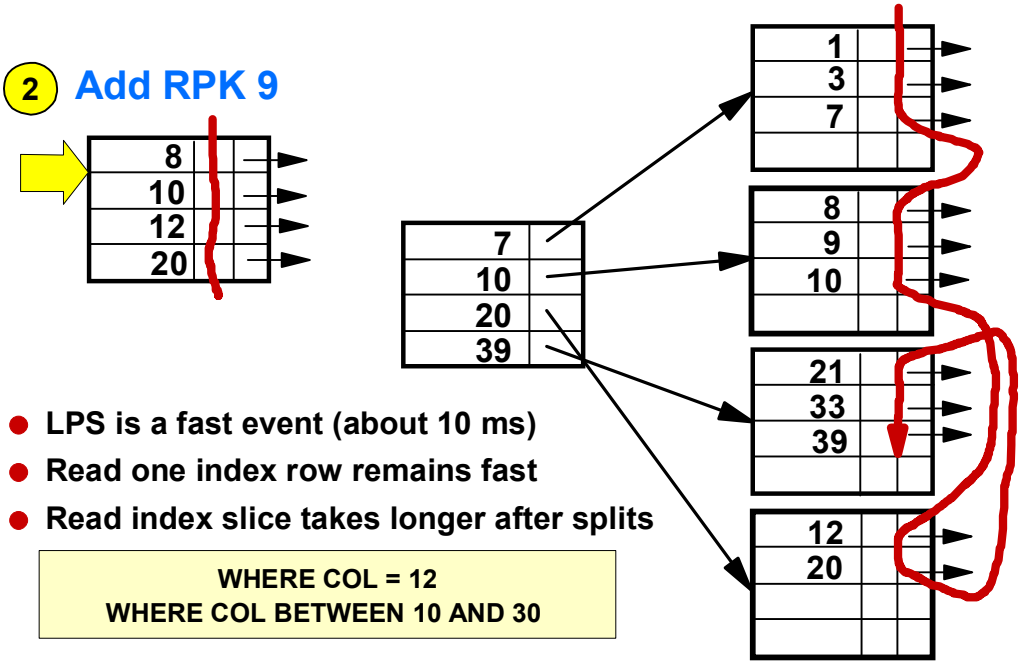
INSERT response time up by about 20 ms (2 TR)

Leaf page split

1.31 Q.31

Leaf Page Split (LPS)

2 Add RPK 9

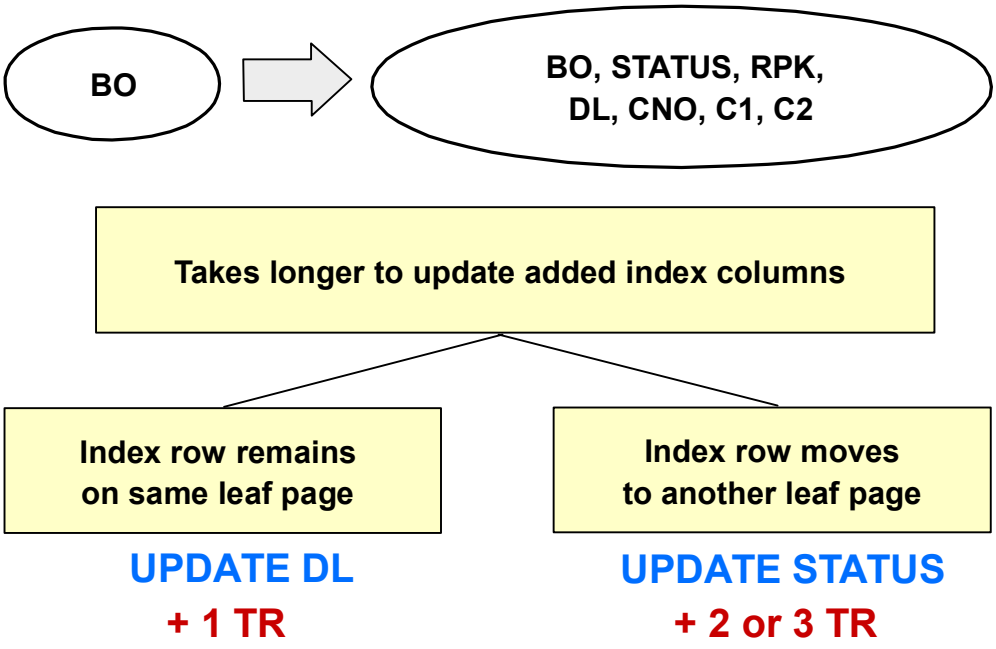


- LPS is a fast event (about 10 ms)
- Read one index row remains fast
- Read index slice takes longer after splits

WHERE COL = 12
WHERE COL BETWEEN 10 AND 30

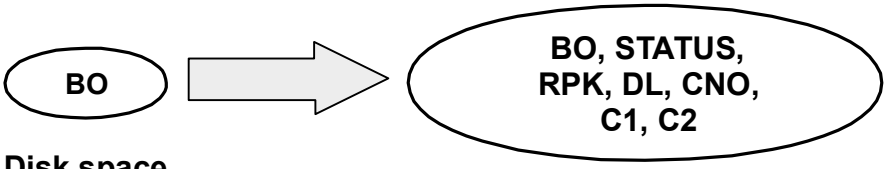
1.32. Q.32

UPDATE With Fat Index



1.33 Q.33

Too Expensive?



- Disk space
- RAM (Non-leaf pages)
- INSERT
- UPDATE
- DELETE
- Index reorg (rebuild)
- Something else?

1,000,000 rows
80 bytes per row

1000 new rows per day
5000 STATUS updates per day

No. The cost very low compared to the **dramatic reduction** in **response time** and **cost** of the common SELECT

The only issue: Index slice read time keeps growing if index reorg interval long (say, a year)

1.35 Q.34

Obsolete Guidelines

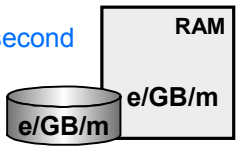
- ~~Do not index volatile columns~~
  STATUS
- ~~Max N indexes per table~~
- ~~etc.~~

Under-indexing: a common mistake

Relevant

- **INSERT and DELETE fast enough after index added?**
  TR = 1 per added or removed index row
- **UPDATE fast enough after index columns added?**
  TR = 1 or 2 per updated index column
- **Index reorg requirement OK?**
  Long index rows (more than 5% of leaf page)
  Hot spots (except end of index)
- **Drive load caused by index maintenance**
  If dozens of random index row inserts or deletes a second
- **Index storage cost**
  < 10 e/GB a month

TR



1.36. Q.35

Happy Users

```
SELECT   DL, STATUS, RPK, CNO, C1, C2
FROM     REQUEST
WHERE    STATUS < 9  AND BO = :BO
ORDER BY DL
```

BO, STATUS, RPK, DL, CNO, C1, C2

1,000,000 rows

F = 1...1000
TR = 1
TS = 0...1000
ET up to 0.1 s

But...even better to materialize 20 result rows at a time?

BO, DL, RPK, STATUS, CNO, C1, C2

1,000,000 rows

F = 1...20
TR = 1
TS = ?

A small change to SELECT?

1.38 Q.36

If SELECT Can Be Changed

BO	DL	RPK	STATUS	CNO	C1	C2
1	18.6.2005	888001	7			
1	18.6.2005	894130	4			
1	19.6.2005	870101	7			
1	20.6.2005	905352	3			
1	20.6.2005	908300	7			
1	15.7.2005	973111	2			
1	31.12.2099	008435	9			
1	31.12.2099	009522	9			

1 %

99 %

Worst input QUBE
with this index
CPU = 1 ms
ET = 10 ms

1.39 Q.37

Index BO: Not Adequate for this SELECT

```
SELECT    DL, STATUS, RPK, CNO, C1, C2
FROM      REQUEST
WHERE     STATUS < 9
          AND
          BO = :BO
ORDER BY DL
```

Who should have seen this?

When?

1.40. Q.38

Systematic Index Design

1. **Detect SELECT statements that are too slow due to inadequate indexing**

   Worst input: Variable values leading to the longest elapsed time

2. **Design indexes that make all SELECT statements fast enough**

   Table maintenance (INSERT, UPDATE, DELETE) must be fast enough as well

* Three Essential Skills

	Before production start	After production start
Detect SELECTs suffering from slow access path	**Quick estimates**	**Exception monitoring**
Design good indexes & make the right choice	● **Minimal-cost semi-fat index** ● **Minimal-cost fat index** ● **Best possible semi-fat index** ● **Best possible fat index**	

More about Designing Indexes in Tapio's course, that I teach for 2-days "Cost saving Database Index Design";
or IBM's course CF961 "DB2 for z/OS Application Performance & Tuning" 5 days;
or the Redbook SG24-7134, "Application Design for High Performance and Availability"

Any questions?

Next, let's discuss JOINs, Star Joins,
Index Reorganization
and Distributed Free Space, and Lock issues

after a short break....

Q.41

Agenda [70 + 70 minutes]:

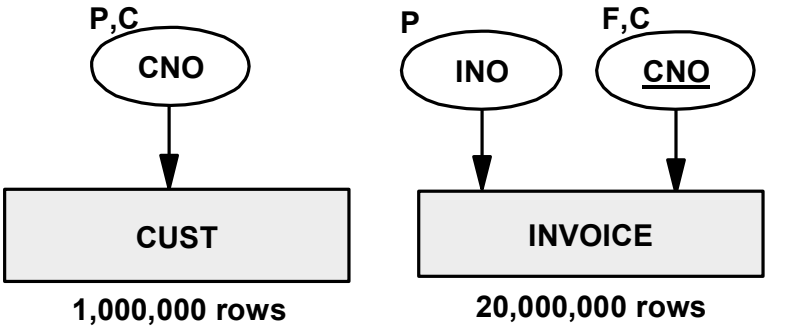
- A "simple" query: a difficult index
- Introduction to Statistics of simple queues
- Effect of queueing on a DASD system
- Query with random "touches"
- Importance of Index improvement
  --break
- **JOIN indexes and JOIN sequence**
- **Star Join Issues & Summary tables**
- **Statistics for Index inserts**
- **Best freespace policy for Indexes**
- **Reorg Index policy for busy systems**
- **A last word: Lock Waits in a Busy System**

J.42

A Fairly Easy Join

```
SELECT  CNAME, IEUR
FROM    CUST, INVOICE
WHERE   CUST.CNO = INVOICE.CNO
        AND
        IEUR > :IEUR
        AND
        IDATE = :IDATE
```

All local predicates refer to one table



How's our indexing?

J.43

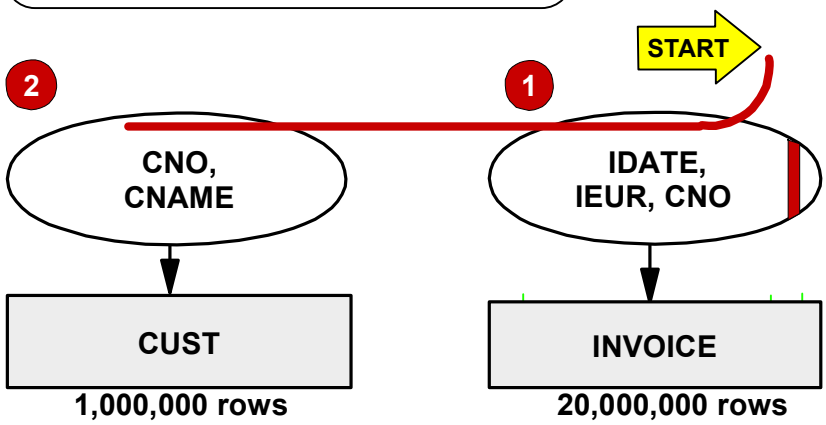
5.02

Ideal Indexes

```
SELECT  CNAME, IEUR
FROM    CUST, INVOICE
WHERE   CUST.CNO = INVOICE.CNO
        AND  IEUR > :IEUR
        AND  IDATE = :IDATE
```

Result up to 200 rows

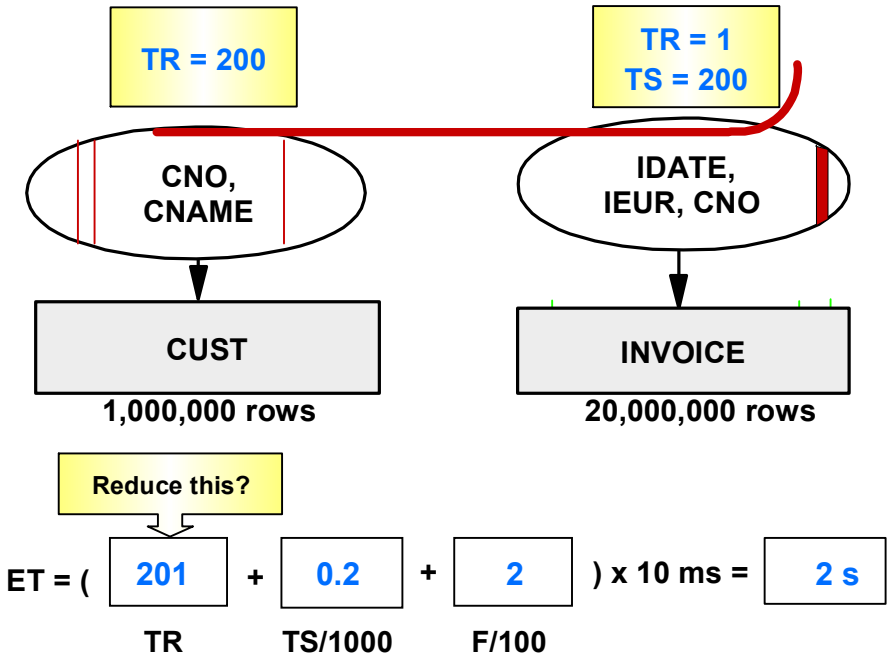
FF up to 0.001%



J.44

5.03

Fairly Fast



5.04 J.45

Which Comes First?

- Indexing affects table access order
- Table access order affects indexing
- What's different about these predicates?

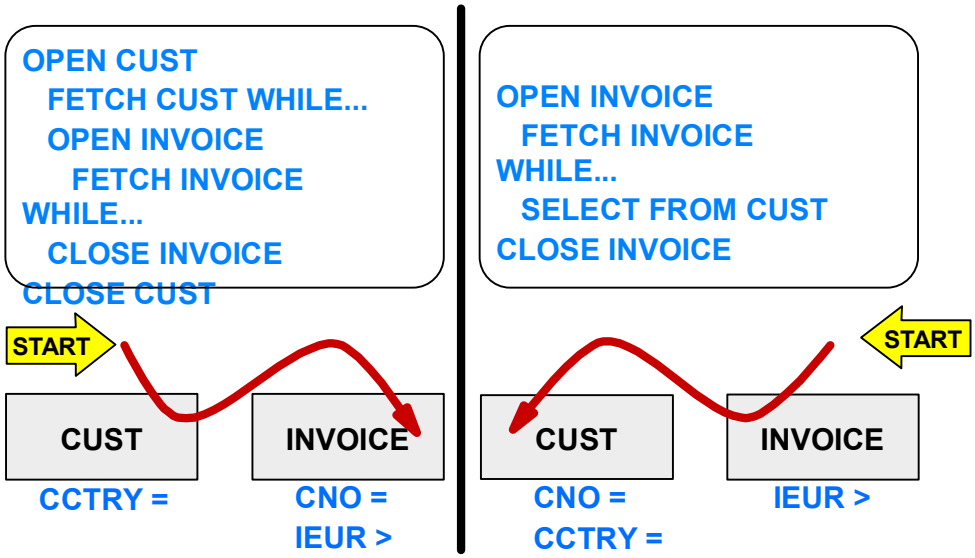
```
SELECT    CNAME, CTYPE, INO, IEUR
FROM      CUST, INVOICE
WHERE     CCTRY = :CCTRY
           AND
          IEUR > :IEUR
           AND
          CUST.CNO = INVOICE.CNO
ORDER BY IEUR DESC
WE WANT 20 ROWS PLEASE
```

FF = 0...10 %

FF = 0.1 %

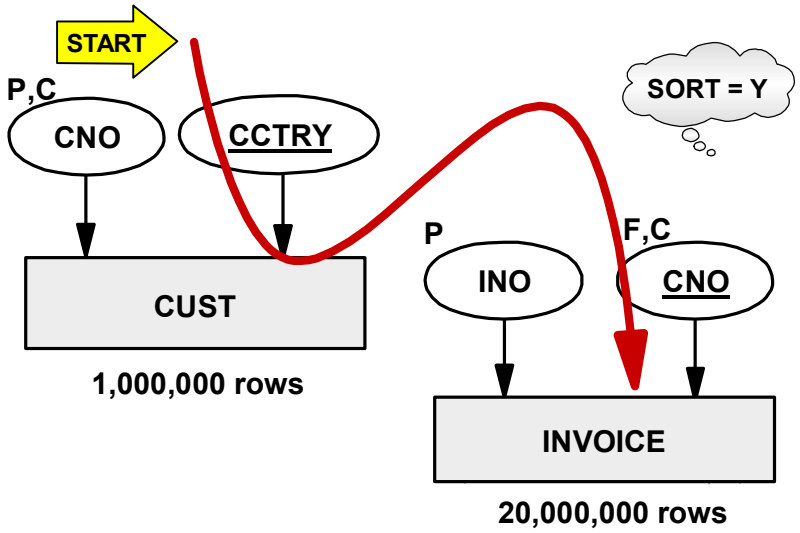
5.05 J.46

Two Cursors: Programmer Decides



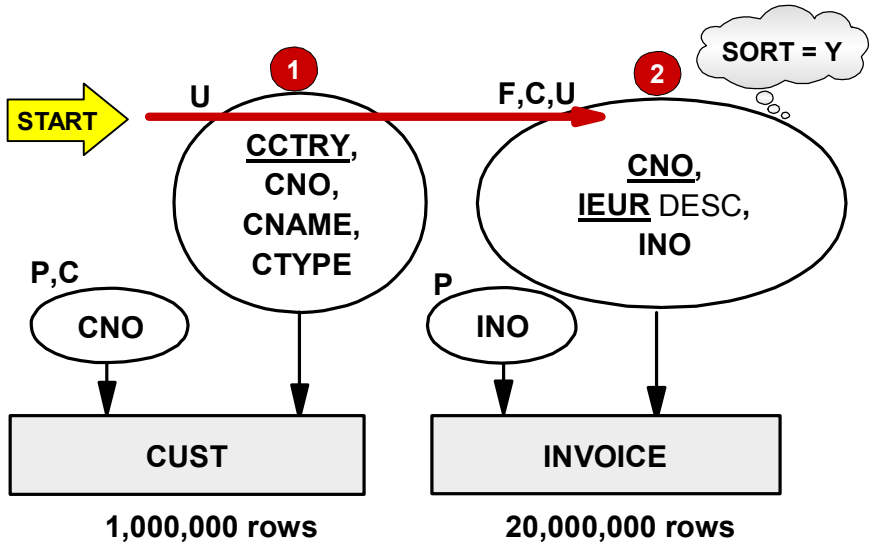
5.06. J.47

V1: Current Indexes



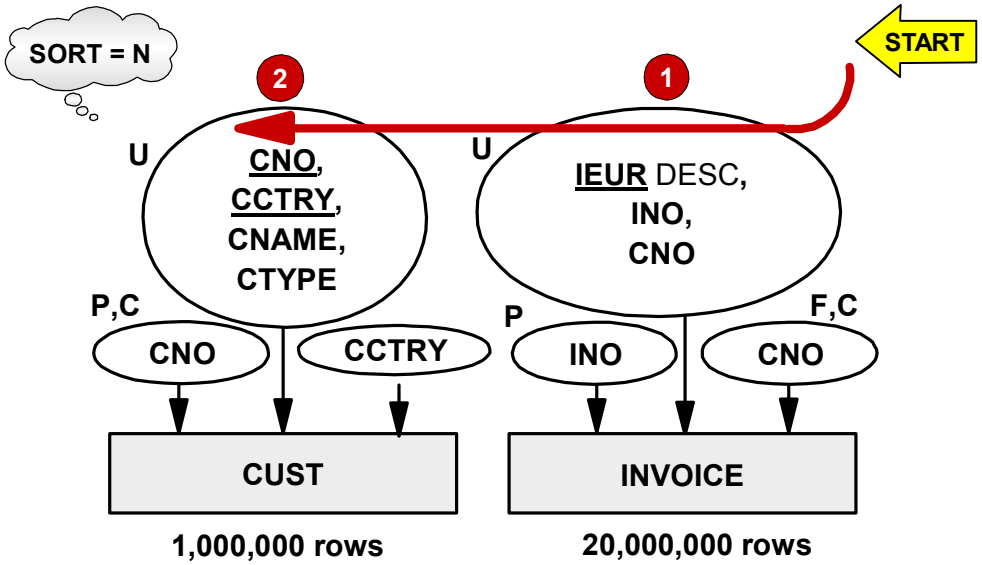
5.07. J.48

V2: Indexes for CUST-Driven Nested Loop



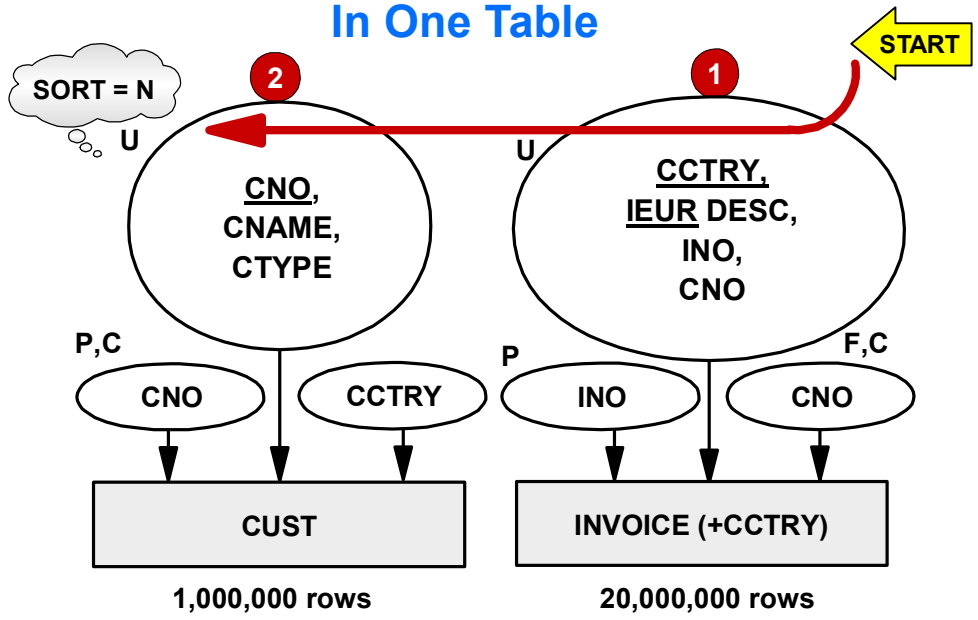
5.08. J.49

V3: Indexes for INVOICE-Driven Nested Loop



5.09. J.50

V4: V3+All Local Predicate Columns In One Table



5.10. J.51

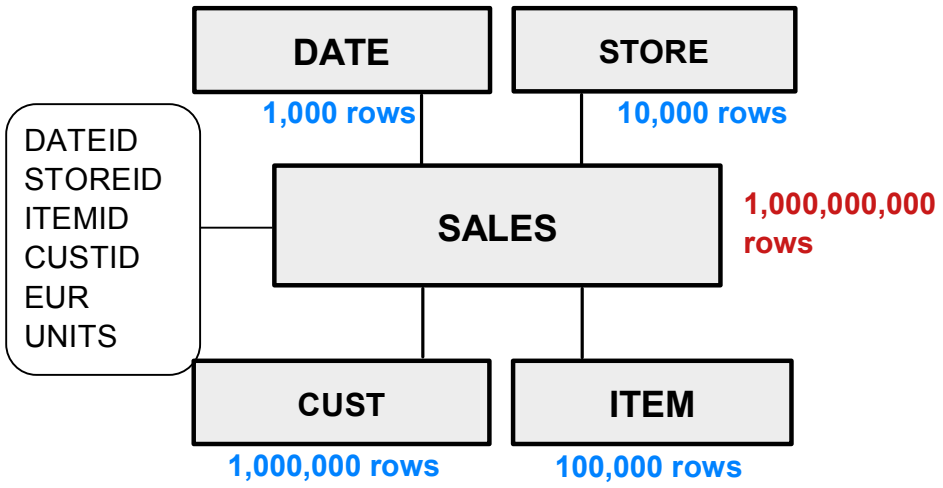
With QUBE you can quickly evaluate alternate JOIN paths with alternate Index Designs

- Clustered or not
- Fat or semi-fat existing index [minimum cost]
- New index, fat or semi-fat
- Denormalized for predicate evaluation
- In one direction or in the other
- Index only or table touches [random ?]

Too long? Too many tables? Avoid JOIN with denormalization and Trigger(s) to support it

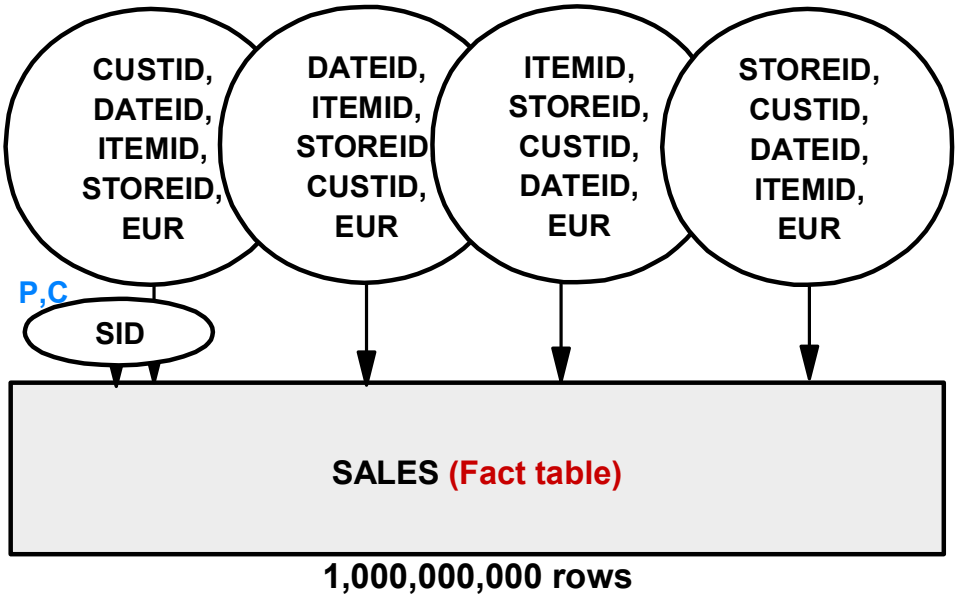
J.52

Star Schema



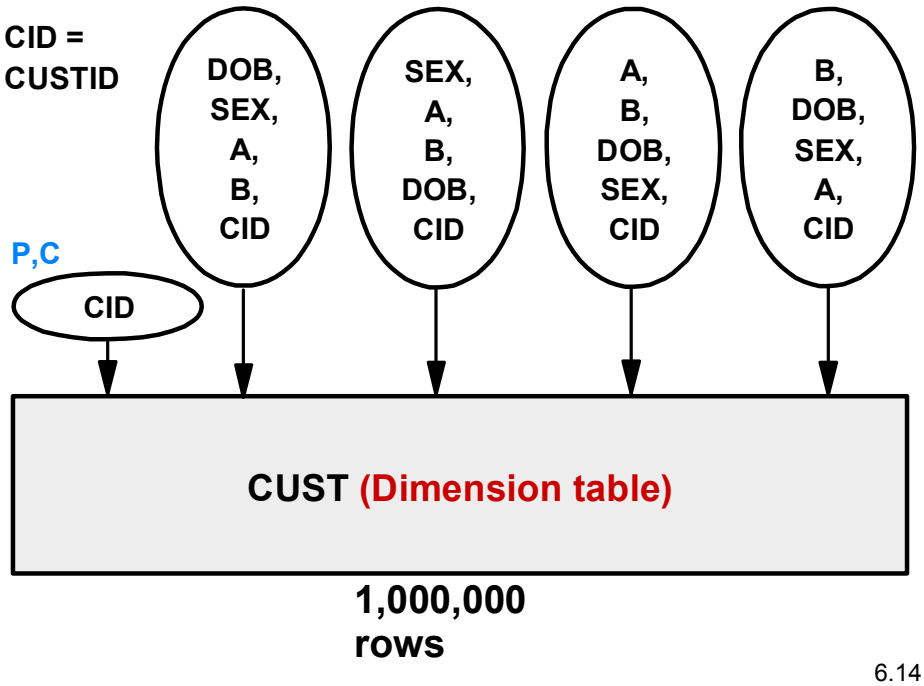
6.12. S.53

One Fat Index per Dimension



6.13 S.54

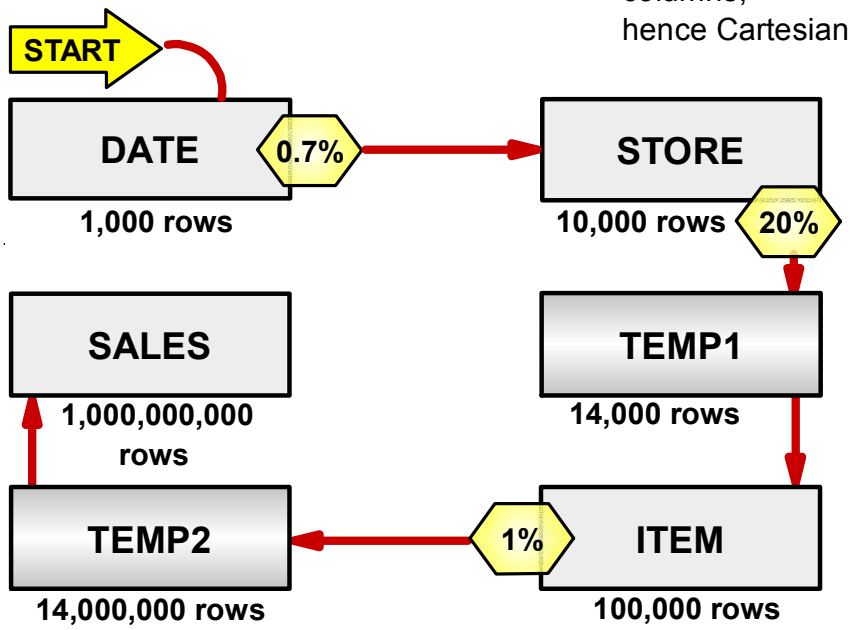
One Fat Index per Predicate Column



6.14 S.55

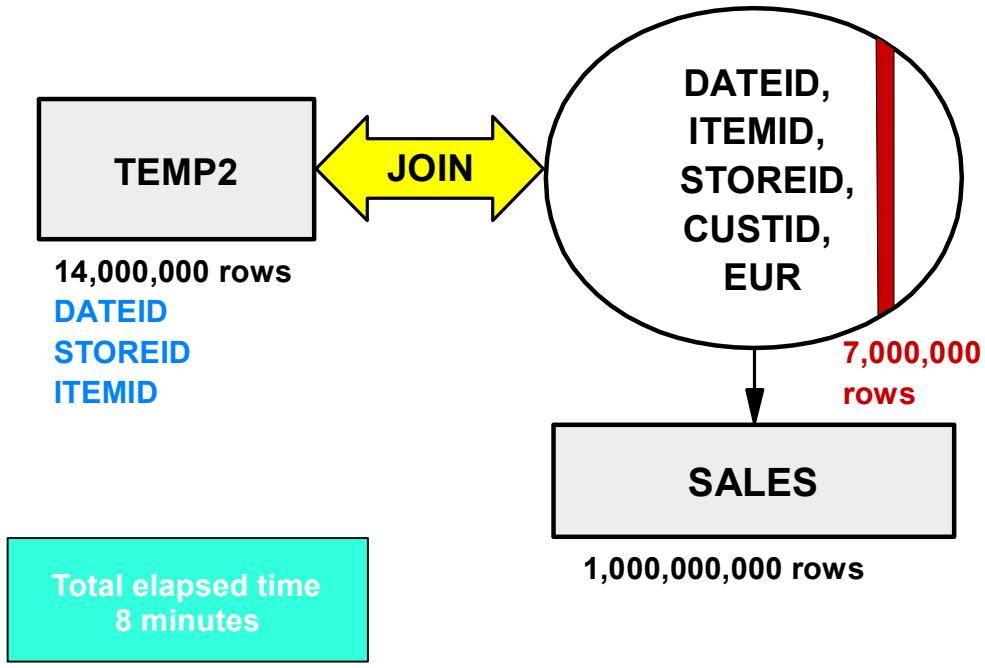
Star Join

The dimension tables have no common columns, hence Cartesian products



6.15 S.56

Last Step



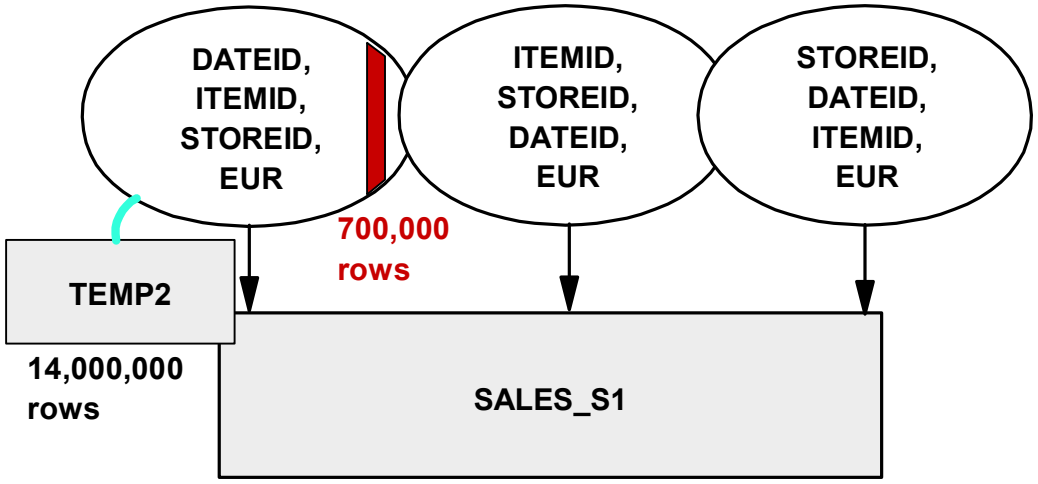
6.17 S.57

If Billion Rows in Fact Table

- Many queries too slow and expensive
  - Even with fat indexes on fact table
  - Final temporary table may get very large (large FFs)
- Standard solution: Summary tables
  - Number of touches down a lot
  - Optimizer should choose right summary table
- Cost of summary tables
  - Daily refresh based on new fact table rows
  - Recreate only if long refresh interval

6.18 S.58

Summary Table 1

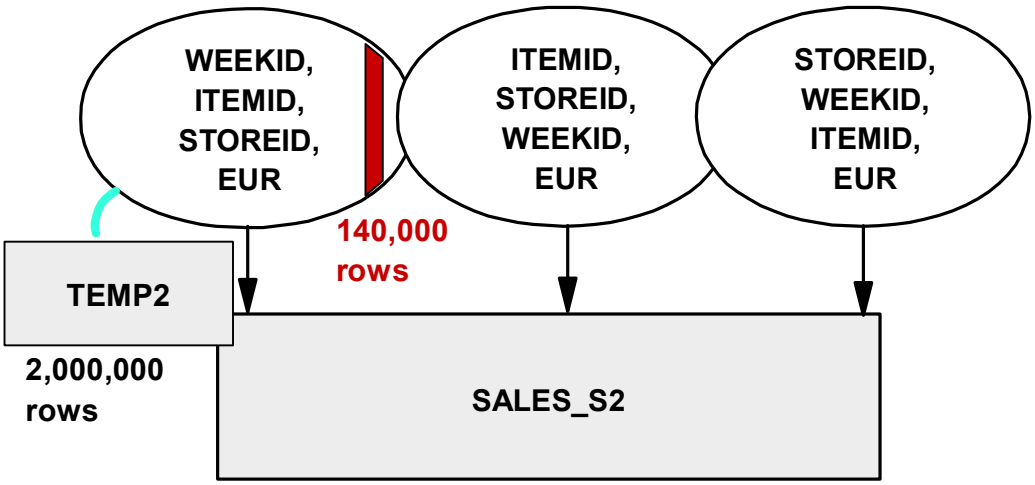


One row per DATEID,ITEMID,STOREID when EUR > 0
Assumption: 100,000,000 rows

Total elapsed time 7 minutes

6.19 S.59

Summary Table 2

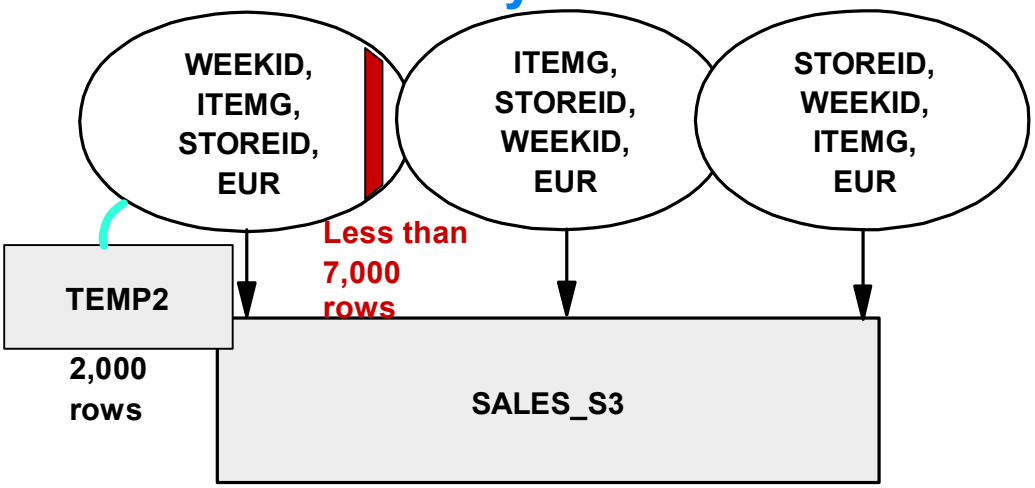


One row per WEEKID,ITEMID,STOREID when EUR > 0
Assumption: 20,000,000 rows

Total elapsed time 1 minute

6.20 S.60

Summary Table 3

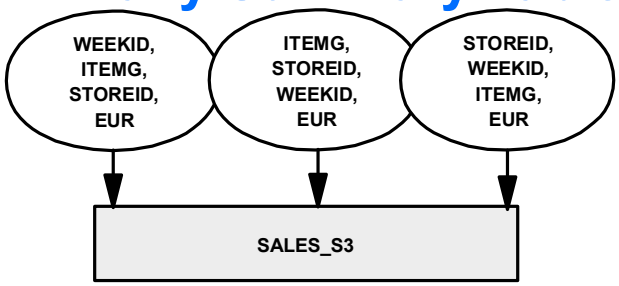


One row per WEEK,ITEMG,STOREID when EUR > 0
Assumption: 1,000,000 rows

Total elapsed time less than 1 s

6.21 S.61

How Many Summary Tables?



Each summary table requires analysis of new fact rows

- Scan 1,000,000 rows (per day)
- Sort 1,000,000 for GROUP BY
- Insert or update summary rows

QUBE: 20 s

Maintaining summary table indexes (random touches)

- Assume 50,000 input rows to SALES_S3 (ITEMG grp ITEMID)
- Synchronous I/O time per index 50,000 x 10 ms = 10 min
- 2 x 10 min = 20 min (touches to index WEEKID localized)

6.22 S.62

Denormalized Fact Table

FATFACT

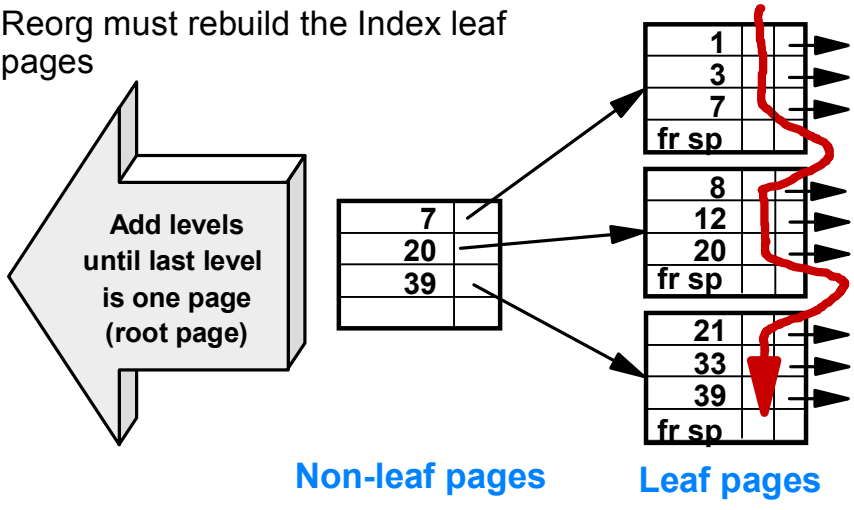
- Consider when less than 100m fact rows
- No joins with dimension tables
  - No Cartesian product
  - Dimension tables used only for referential integrity
- Excellent indexability
  - All predicate columns in one table
  - Example query: (WEEK,ITEMG,STOREG,EUR)
- Reduces need for summary tables
  - But summary table indexes are smaller
- Adding dimension attributes expensive
  - Read all fact rows & write back
  - Enormous disk drive load (lots of writes)

S.63

6.23

Index in Perfect Physical Order

Reorg must rebuild the Index leaf pages



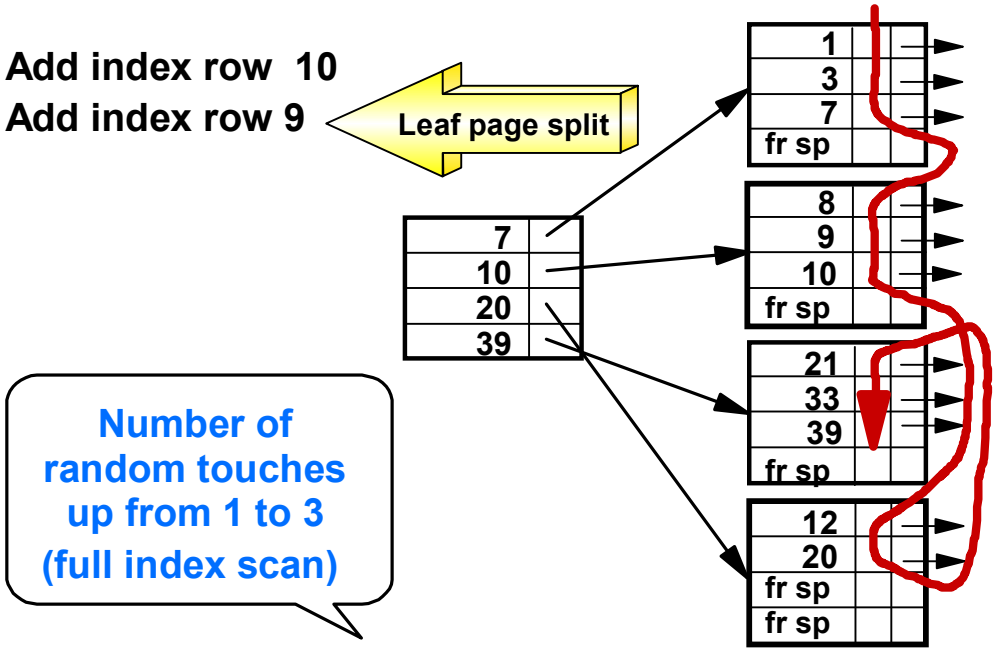
For example, 1,000,000 rows need 50,000 leaf pages

7.02..

R.64

Index Not Perfect Any Longer

Add index row 10
Add index row 9

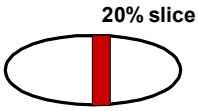


Number of random touches up from 1 to 3 (full index scan)

7.03.. R.65

Leaf Page Split Ratio (LPSR)

I/O time after leaf page splits
= (1+ (LPSR x A/B)) x ORIG



A = I/O time per random read
B = I/O time per sequentially read page
LPSR = N of splits / N of leaf pages
ORIG = I/O time before leaf page splits

Example
A = 10 ms
B = 0.1 ms [A/B=100]
LPSR = 0.01
ORIG = 1 s (10,000 leaf pages)
I/O time after splits
= (1+ (0.01 x [100])) x 1s = 2s

Assumptions

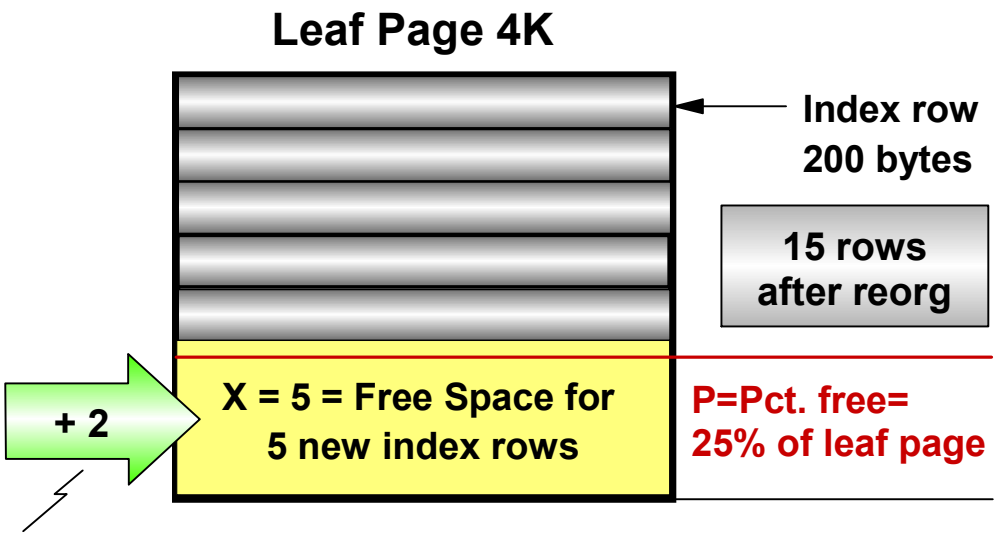
Seq I/O time
= NLP x B
(The random I/O
to first page ignored)

Random I/O time
= LSPR x NLP x A
(one random I/O
per leaf page split)

7.04. R.66

Index has Grown by 13% After Reorg

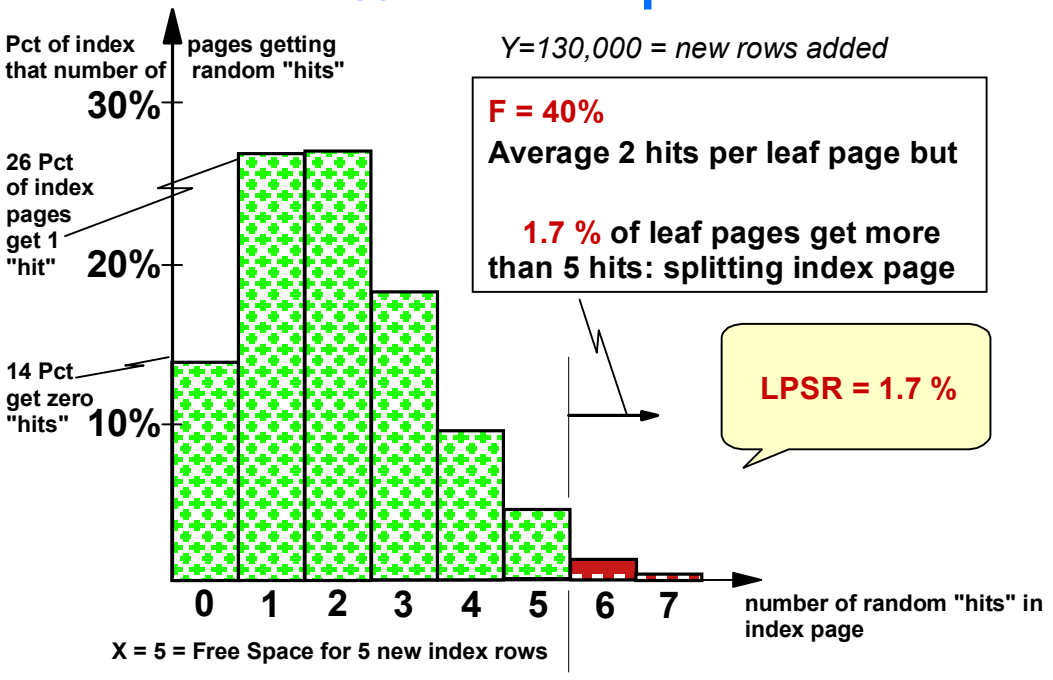
Ex: Y=130,000 = new rows added to table, random inserts



15 rows/pg x 0.13 growth = avg of 2 rows/pg added

7.05. R.67

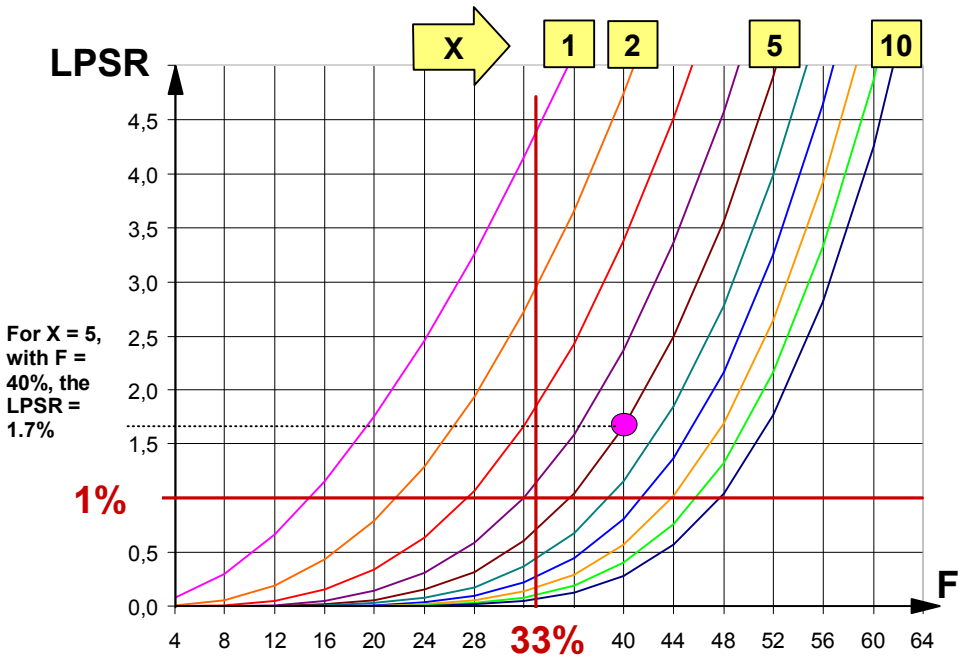
When 40% of Free Space Used



F = percent of the distributed free space used up at a certain point in time

7.06. R.68

LPSR vs F for X rows of index free space



X = 5 = Free Space for 5 new index rows reaches LPSR=1% when F=36%

F = percent of the distributed free space used up at a certain point in time

7.07. R.69

Insert Pattern

- **If new index rows spread randomly on leaf pages**
  - Choose P based on row length and index growth
  - Objective: Leaf Page Split Ratio (LPSR) < 1%

Next pages

- **If ever-increasing index key**
  - P = 0 (fixed length) or P = 20 (variable length)
  - UPDATE = DELETE + INSERT (index row may move)
- **If many new index rows to a small area (hot spot)**
  - Tailored solution
  - Consider resident index

7.08 R.70

Short Index Rows < 2%

4K: < 80 bytes 8K: < 160 bytes

Basic recommendation

P = 10%
Reorganize index when it has grown by 5%

If longer reorganization interval is required

P = 20%
Reorganize index when it has grown by 12%

LPSR < 1% with random inserts

7.10 R.71

Medium Index Rows 2...5%

4K: 80...200 bytes 8K:160...400 bytes

Basic recommendation

X = 5
Reorganize index when 1/3 of free space filled
When index growth G = 33 x (P/(100-P)) %
See next page

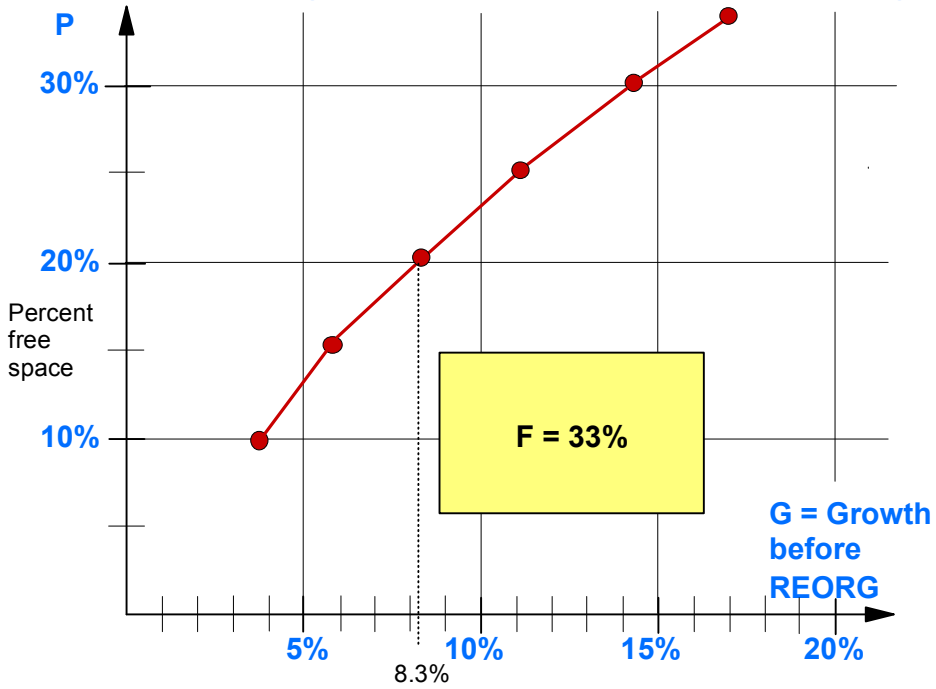
Example

Index row 200 bytes, leaf page 4K
5 x 200 bytes = 1,000 bytes: P = 25
Reorganize index when it has grown by 11%

LPSR < 1% with random inserts

7.11. R.72

Calculating Row Growth before Reorg

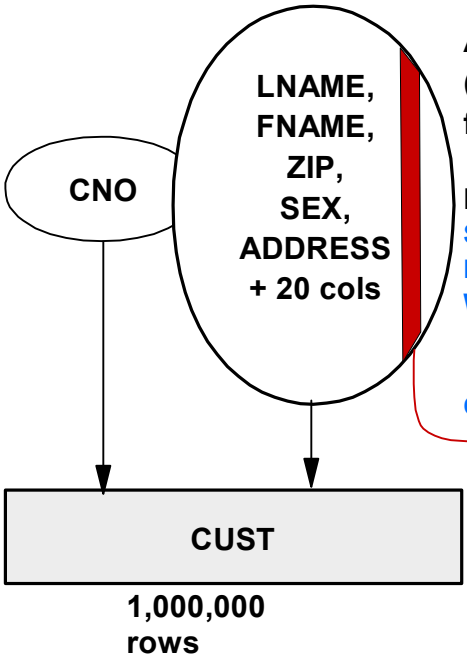


F = percent of the distributed free space used up at a certain point in time

7.12 R.73

Long Index Row > 5%

4K: > 200 bytes 8K: > 400 bytes



A good fat index
(but not the ideal index)
for many SELECTs

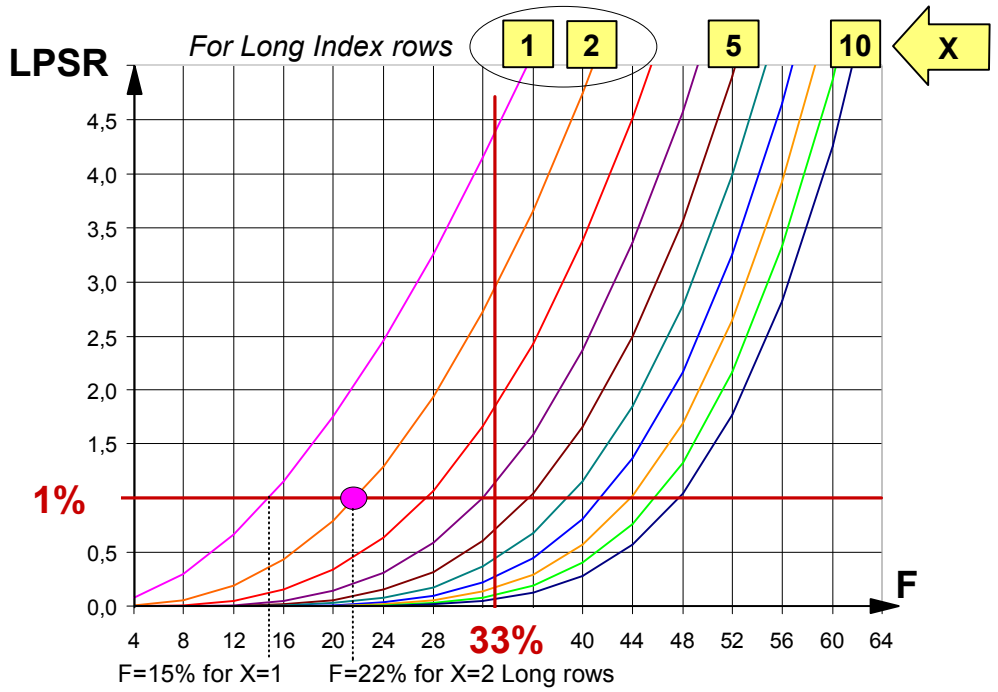
Example:

```
SELECT 10 columns
FROM   CUST
WHERE  LNAME BETWEEN :hv1 AND :hv2
         AND
         SEX = 'M'
ORDER BY LNAME, FNAME
```

Worst case:
100,000 x 500 bytes

7.13 R.74

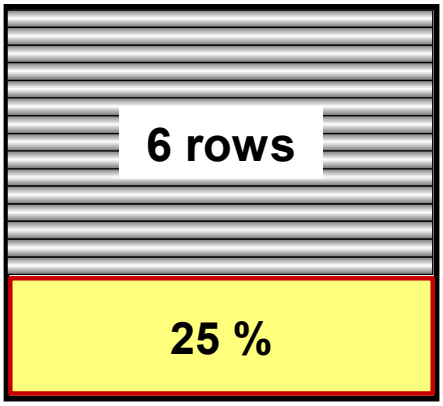
LPSR vs F for X rows of index free space



F = percent of the distributed free space used up at a certain point in time

7.14. R.75

For 8 rows/pg Choose X = 2



LPSR = 1 when F = 22%

F = 22% when 0.44 new rows/LP

Index growth = 0.44/6 = 7%

I/O time for 100K row slice

(100,000 r / [6 r/pg]) x 0.1 ms = 1.7 s

+7%

(1+1) x 1.7 s = 3.4 s

7.15. R.76

What's the effect of a busy system on Lock-waits? [see pg.7]

- All syncronous reads [TR's] tend to take longer
  [recall the u/(1-u) factor]
- CPU queuing may also increase LW-time
- Better indexes reduce the number of TR's and
  tend to reduce the drive load [lower the "u"]
- Better indexes shorten the Elapsed time and reduce
  conflicting lock waits, timeouts and deadlocks
- New indexes or fat index updates add to X-lock time
  but pre-reads can dramatically reduce
  X-lock time even with more ix's!

If 2 lock requests occur on average per second for an object
and it is locked on average 0.2 sec
then it is "busy" 40% [u=0.4]; { goal: less than 10% busy}
lock wait is then Q = .4/.6 x .2 = .13 sec waiting for a lock

L.77

Any Questions?

Learn more and practice proven techniques....

Next class 10/12-13/2006 Columbia MD

Cost-Saving Database Index Design

Presented by Larry Kintisch, ABLE Information Services

Tapio Lahdenmäki, Author
Independent consultant Database performance

tapio.lahdenmaki@siol.net

Thank you for your participation! This course is copyrighted by Tapio Lahdenmäki, S.P

Do not copy or transmit electronically without the permission of the author at
www.Tapio1.com

Course schedules outside the USA are at the website above. For USA classes see
www.DBIndexDesign.com

Please send questions, comments and reports of your successes with these methods to
Larry@DBIndexDesign.com
ABLE Information Services 845-353-0885
Q913-BWDB2UG.prz