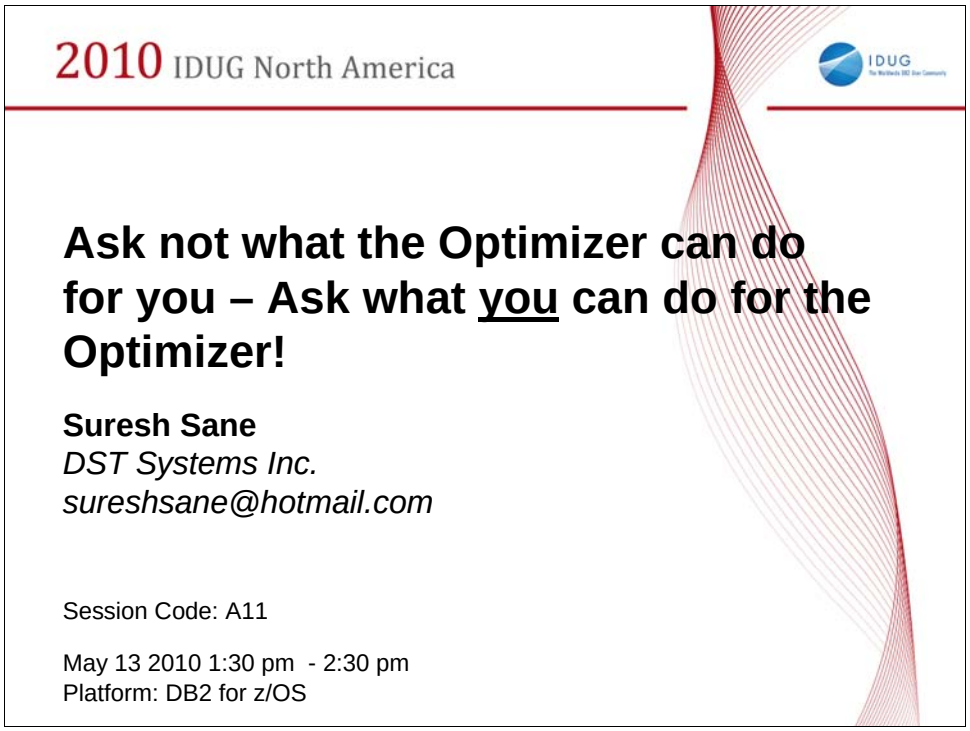


"This query used to run in 5 seconds and now takes an hour! How could the Optimizer do this to me?" Sounds familiar? But are you sure the Optimizer is at fault?

In this session, we will examine the challenges faced by cost-based optimization and identify steps to minimize, if not eliminate, such behavior. We will also cover various ways of influencing and stabilizing the optimal access path.

We will transition from the "Why doesn't IBM?" complaint to the more positive "What can I do TODAY?" action plan.

Session Outline

1. **Cost-based optimizer basics**
2. **Stats, stats and even more stats**
3. **Case studies**
4. **Access path management - Influencing, stabilizing and falling back**
5. **What does the future hold?**

2

Cost-based optimizer basics

- •Filter factors
- •Issues in access path creation

Stats, stats and even more stats

- •Basic stats
- •NUD
- •Correlation
- •Histogram

Case studies

- •Case 1 – Who's on first? (Basic stats)
- •Case 2 – Filtering matters! (Basic stats)
- •Case 3 – Averages can be deceiving! (NUD stats)
- •Case 4 – Are we related? (Correlation stats)
- •Case 5 – Ranges are skewed too! (Histogram stats)
- •Case 6 - The "Pick me!" index (Index design)

Access path management - Influencing, stabilizing and falling back

- •Hints
- •Stats
- •Tricks
- •Package stability

What does the future hold?

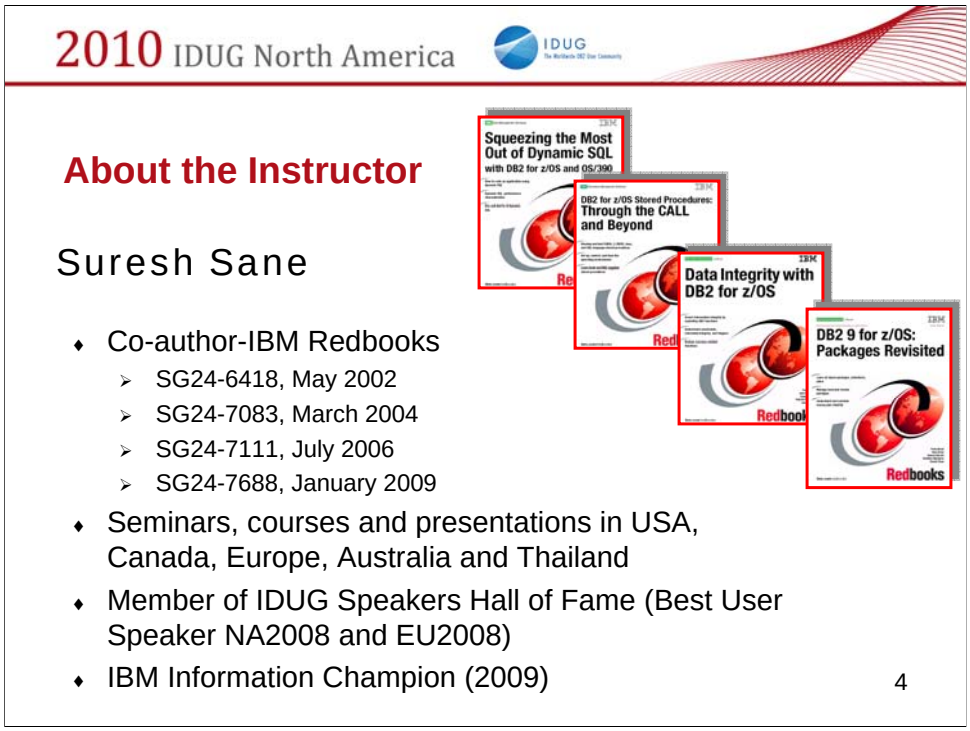
- •Directions
- •My wish list

Warning!

- 94 slides, oh my! (yes, slipped past my Thread Chair!)
- I do not intend to cover each one – several, which show the detail or are basic in nature, will be skipped.

3

Actually…my thread chair (Bryan Paulsen) did catch it, but decided to let it go… "Thread chair did notice this was 94 slides and is reminding you of the one hour session time."



Suresh Sane is an IDUG Hall of Fame speaker with two Best User Speaker awards and numerous top 10 finishes. He has lectured worldwide and co-authored 4 IBM Redbooks (Dynamic SQL, Stored Procedures, Data Integrity and DB2 Packages). He was recognized as an IBM Information Champion in 2009.

He is a long time IDUG volunteer and was the Conference Chair for IDUG NA 2008. He currently serves on the IDUG Board of Directors.

Contact Information:

sssane@dstsystems.com

Suresh Sane

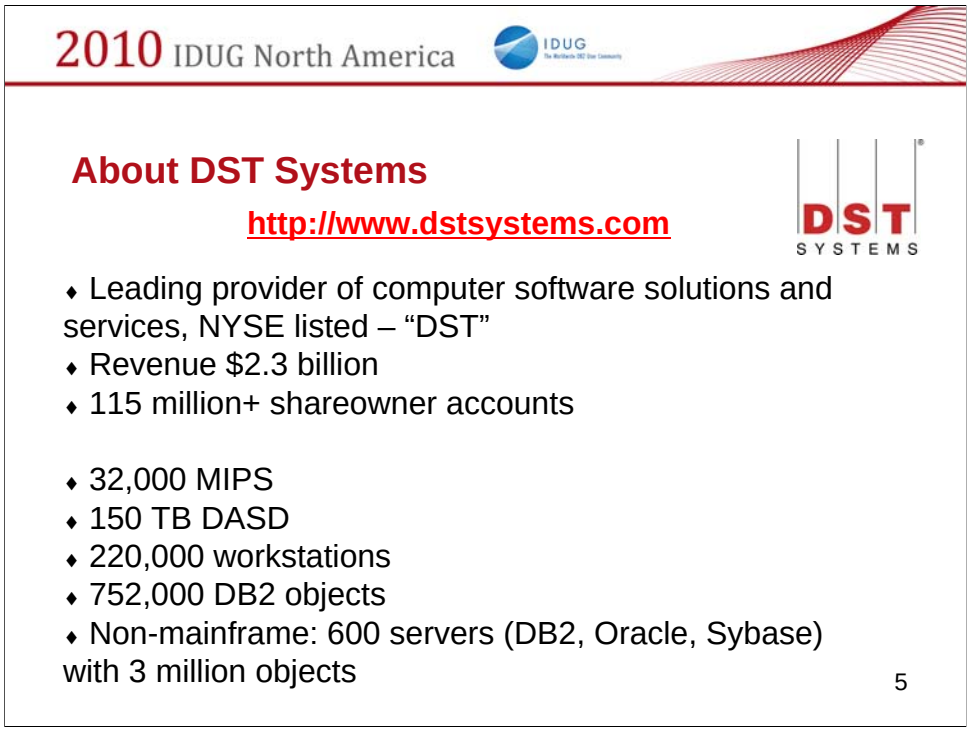
DST Systems, Inc.

1055 Broadway

Kansas City, MO 64105

USA

(816) 435-3803



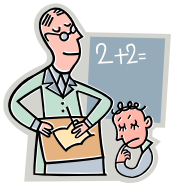
If you have ever invested in a mutual fund, have had a prescription filled, or are a cable or satellite television subscriber, you may have already had dealings with our company.

DST Systems, Inc. is a publicly traded company (NYSE: DST) with headquarters in Kansas City, MO. Founded in 1969, it employs about 10,000 associates domestically and internationally.

The three operating segments - Financial Services, Output Solutions and Customer Management - are further enhanced by DST's advanced technology and e-commerce solutions.

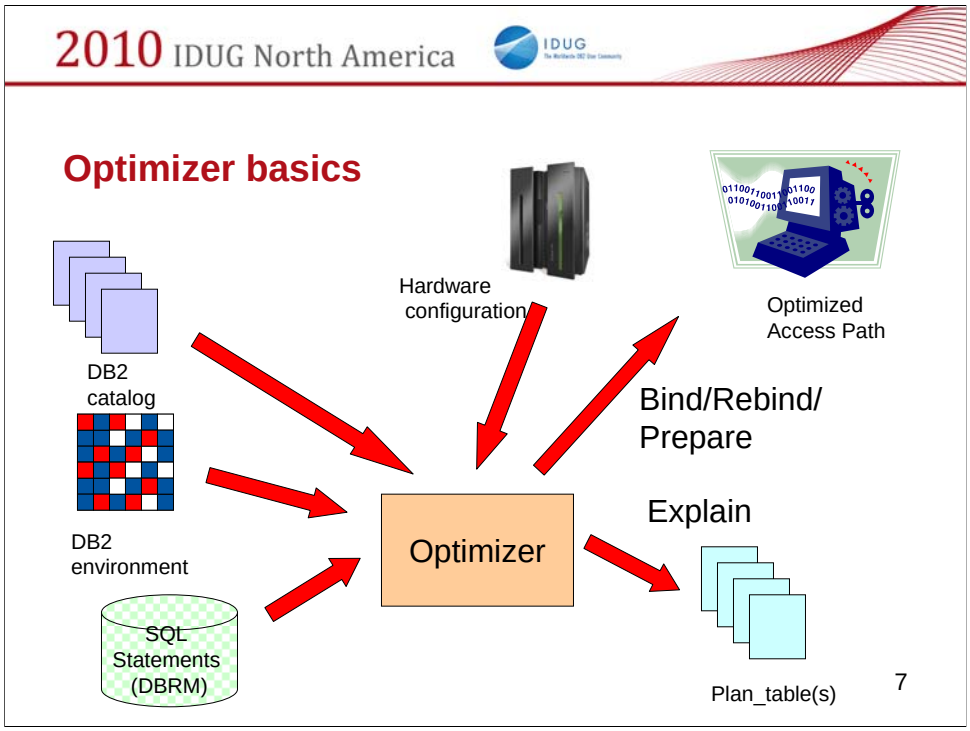
Where Are We?

1. **Cost-based optimizer basics**
2. **Stats, stats and even more stats**
3. **Case studies**
4. **Access path management - Influencing, stabilizing and falling back**
5. **What does the future hold?**

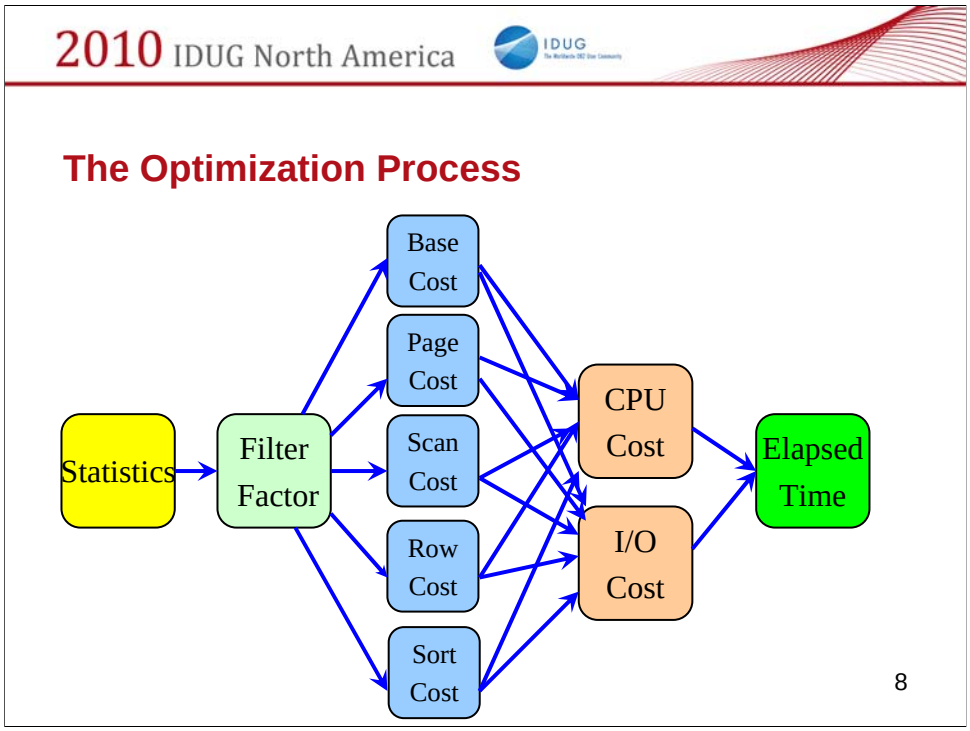


6

In this section we will cover the basics and explain how the optimizer determines the optimal access path.



The basics of how the optimizer creates an access path and externalizes it in a plan_table(s) when using the EXPLAIN option.



Some of the considerations used by the optimizer for access path selection.

The basic steps of the optimization process. It is easy to underestimate the magnitude of this task. The number of access paths which are theoretically possible increases exponentially as the number of tables and indexes increase..

To me, it is not surprising that it picks a less than optimal access path in rare cases. What is surprising is that it picks the right access path almost all the time (97% or better according to one estimate)!

Simple Filter Factor Example

$$\text{Filter Factor} = \frac{\text{Number of result rows}}{\text{Number of source rows}}$$

```
SELECT …*
WHERE     LAST_NAME = ?
AND          DEPT = ?
```

- Calculate the filter factor:
  - 1/COLCARD of LAST_NAME
  - 1/1000 = .001
  - 1/COLCARD of DEPT
  - 1/10 = .1
- AND (multiply) the predicates together:
  - .001 * .1 = .0001

9

Calculations the optimizer uses for combining filter factors.

Effect of clustering

- Filtering using an index affects the number of data **rows** that must be accessed (assuming no index-only access)
- Clustering has a huge impact on the number of data **pages** that must be accessed
- Example – 50 rows using a non-clustering index could mean 50 getpages while 100 rows using a clustering index could mean just 1 or 2 getpages

10

Clustering has to be taken into account since a clustered index matches the order of data in the table and can result in a far fewer page "touches" – a concept we will explore further later on.

Literals and REOPT

- With host variables for static SQL and parameter markers for dynamic SQL, the exact value at bind/prepare time is not known. REOPT allows the creation of an access path at run time
  - REOPT (NONE) – default
  - REOPT (ALWAYS) – for static and dynamic SQL
  - REOPT (ONCE) – for dynamic SQL
  - REOPT(AUTO) – DB2 9 for dynamic SQL
- Specifying literals
  - For static, better performance but hard coding may necessitate SQL changes when business needs change
  - For dynamic, better access path but may lead to no cache reuse

11

Effect of working with an unknown value (host variable or parameter marker) vs. a known value (a literal).

Where Are We?

1. Cost-based optimizer basics
2. Stats, stats and even more stats
3. Case studies
4. Access path management - Influencing, stabilizing and falling back
5. What does the future hold?

12

In this section, we will cover what stats can be gathered with the RUNSTATS utility and the scenarios where they are useful.

I will cover this section quickly, emphasizing a few key issues. I expect most of you are already familiar with the basics and the discussion is somewhat dry.

Why stats?

"Measure what is measurable, and make measurable what is not so." - **Galileo Galilei**

"Not everything that can be counted counts, and not everything that counts can be counted" - **Albert Einstein**

"There are three kinds of lies: lies, damned lies, and statistics." - **Benjamin Disraeli**

… and more to the point.. for DB2:

"What I mean by 'all optimizers are hungry'… We push collection of statistics so the optimizer more accurately estimates the cost … The better we estimate the available choices, the better able we are to choose the cheapest one." - **Patrick Bossman, IBM, on IDUG DB2-L June 26, 2007**

13

Why should you care about stats?

Table statistics

- CARDF
  - Number of rows in the table or partition
- NPAGESF
  - Number of pages containing data
- NACTIVEF
  - Number of active pages in the tablespace
  - Only used for single table simple tablespaces
- PCTROWCOMP
  - Percentage of compressed rows

14

Some of the critical catalog statistics for tables and tablespaces. Notice that the older columns in the table (CARD, NPAGES etc) have now been obsoleted.

Index statistics

- NLEAF
  - Number of active leaf pages
- NLEVELS
  - PROGRAM_NME PLATFORM_ID
- CLUSTERRATIOF
  - Percentage of rows in clustering order
- CLUSTERING
  - Is the index the clustering index?
- FIRSTKEYCARDF
  - Number of distinct values for the first index column
- FULLKEYCARDF
  - Number of distinct values for the full index key

15

Important catalog statistics for the index.

Non-uniform distribution and correlation

- Non-uniform distribution (data skew)
  - Data is not uniformly distributed
    - Point-skewed (e.g. active = 95%, retired = 5%)
    - Range-skewed (e.g. salary range is 30K to 300K, with 90% earning less than 100K) – see Histogram statistics
- Correlation
  - Data in two or more columns is not independent
  - Affects how filter factors are multiplied
  - E.g. State and zip code (64105 exists only in MO) or make and model (Civic is made only by Honda) – 100% correlation

16

Explaining the concept of non-uniform distribution and correlation.

Selectivity statistics

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

17

Catalog statistics that determine the selectivity of predicates. We will discuss each of these in detail in the following slides.

Single column cardinality

- Number of distinct values for a column
- Assumes uniform distribution
- Stored as
  - SYSCOLUMNS.COLCARDF
  - SYSINDEXES.FIRSTKEYCARDF
- Used when
  - Host variables, parameter markers, special registers etc
  - No other stats available
- Collected by issuing:
  - RUNSTATS TABLESPACE(..) TABLE (..) COLUMN(..)
  - RUNSTATS INDEX(..)
  - RUNSTATS TABLESPACE(..) INDEX(ALL)

18

Stats that can be gathered for a single column.

HIGH2KEY/LOW2KEY

- Stored as SYSCOLUMNS.LOW2KEY, HIGH2KEY
- Used when
  - Interpolation for range predicates
  - LIKE, BETWEEN, <, <=, >, >=
  - Literal value is known (not for host variables)
- Can be used in combination with single column frequencies for more accurate estimates
- Histogram stats covered later

19

Somewhat primitive measures (useful nonetheless) are the LOW2KEY and HIGH2KEY. Finer histograms are the way of the future that make these range stats more accurate.

Single column frequency

- Stored as SYSCOLDIST.FREQUENCYF (TYPE = F, NUMCOLS = 1)
- Information about non-uniform distribution (data skew)
- Used when
  - Literal value is known (not for host variables)
  - EQUAL, IS NULL, IS NOT NULL
  - LIKE, BETWEEN, <, <=, >, >=
- Collected by issuing:
  - RUNSTATS INDEX (..) COLUMNS(..) – leading cols
  - RUNSTATS INDEX(.. FREQVAL NUMCOLS(..) COUNT(..) – how many columns and how many values
  - V8 allows this for any columns with the COLGROUP keyword

20

Keep in mind the COLGROUP keyword introduce in V8 which makes this more powerful.

Multi-column cardinality

- Stored as
  - SYSINDEXES.FULLKEYCARDF
  - SYSCOLDIST.CARDF (TYPE=C, NUMCOLS>1)
- Assumes uniform distribution
- Used when
  - Mainly intended for columns of an index
  - Does not need literals (can be used with host variables or registers)
  - Supports multi-column frequencies (when collected)
  - Helps with multi-column join predicates
- Collected by issuing:
  - RUNSTATS INDEX (.. KEYCARD) – groups of leading cols e.g. C1, C1+C2, C1+C2+C3, C1+C2+C3+C4
  - COLGROUP instead of KEYCARD allows this for any columns – not just those in an index

21

Keep in mind the COLGROUP keyword introduced in V8 which makes this more powerful.

Multi-column frequency

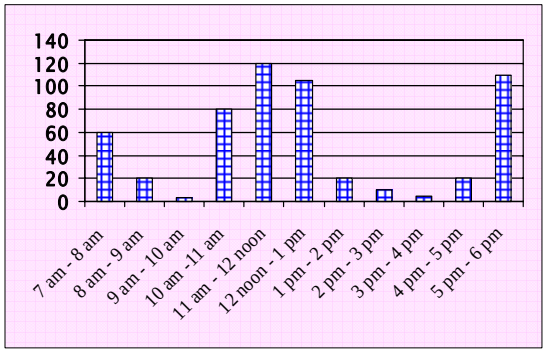
- Similar to single column frequencies
- Identifies non-uniform distribution (data skew) for groups of columns
- Stored as
  - SYSCOLDIST.FREQUENCYF (TYPE=F, NUMCOLS >1)
- Collected by issuing:
  - RUNSTATS INDEX (..) COLUMNS(..) – leading cols
  - RUNSTATS INDEX(.. FREQVAL NUMCOLS(..) COUNT(..) – how many columns and how many values
  - COLGROUP instead of KEYCARD allows this for any columns – not just those in an index

22

Keep in mind the COLGROUP keyword introduced in V8 which makes this more powerful.

Histogram statistics

- Histogram statistics enable DB2 to improve access path selection by estimating predicate selectivity for **range** predicates (thru V8 skew only for **single** values).
- Example - How many customers between 11 am and 1 pm?



23

Example of how the additional stats help the Optimizer.

Histogram statistics - RUNSTATS

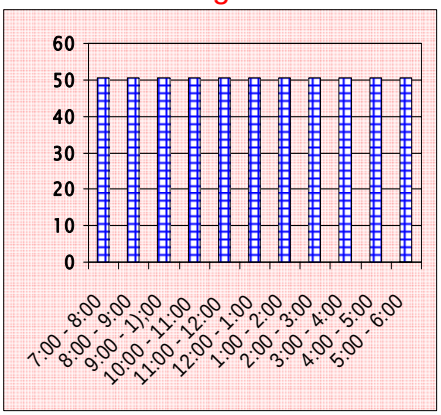
- Collected by issuing:
  - RUNSTATS INDEX(…) HISTOGRAM NUMCOLS ..
- Max 100 quantiles for a column.
- A column value not broken across quantiles.
- Quantiles of similar size but:
  - Tries to avoid big gaps between quantiles.
  - NULL always has a separate quantile.
  - *Height-balanced* (as opposed to *Width-balanced*) – i.e. each interval (range) will have about the same number of rows (not the same number of values) – see next slide.
- If < 10 distinct values, reverts to distribution stats.
- Not supported for LOAD or REORG.
- Possible for column groups as well as single columns.
- Composite indexes with mixed order (ASC/DESC), only possible for prefix columns of the same order.

24

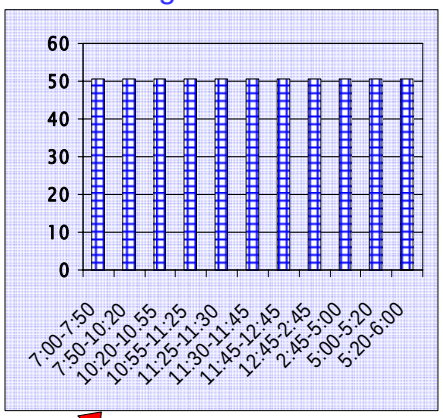
Details of what stats are collected.

Histogram statistics

Without Histogram:



With Histogram:



Note different widths

25

Note – these are equal-height, not equal-width.

The graph at the left is a visual representation of what the optimizer "sees" – even though the catalog does not contain such slices of data.

General recommendations for RUNSTATS

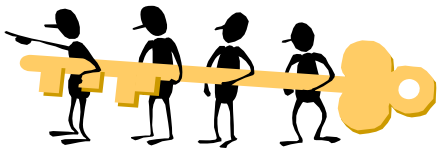
- Collect basic stats (table and index) always.
- Focus on columns in WHERE clauses and in ORDER BY clauses (DSN_PREDICAT table of extended explain).
- Collect non-uniform distribution stats when data skew is present.
- Collect correlation stats when two or more columns are highly correlated.
- Collect histogram stats when range skew and range predicates exist.
- Remember how “hungry” the Optimizer is!

26

General advice on what stats should be collected.

Where Are We?

1. Cost-based optimizer basics
2. Stats, stats and even more stats
3. Case studies
4. Access path management - Influencing, stabilizing and falling back
5. What does the future hold?



27

In this section, we will cover 6 examples. We will begin with a simple example to show why basic stats are necessary. We will build on this to explore the impact of non-uniform distribution , correlation and range skew. Finally, we will cover how an “irresistible” index – one good for all seasons – can be created.

Keep in mind that the cases discussed here are sterile in nature – real life scenarios are far more complex. They are intended to illustrate the need for accurate and detailed stats and to show how a “perfect” index can be discovered.

Case studies

- Case 1 – Who's on first? (Basic stats)
- Case 2 – Filtering matters! (Basic stats)
- Case 3 – Averages can be deceiving! (NUD stats)
- Case 4 – Are we related? (Correlation stats)
- Case 5 – Ranges are skewed too! (Histogram stats)
- Case 6 - The "Pick me!" index (Index design)

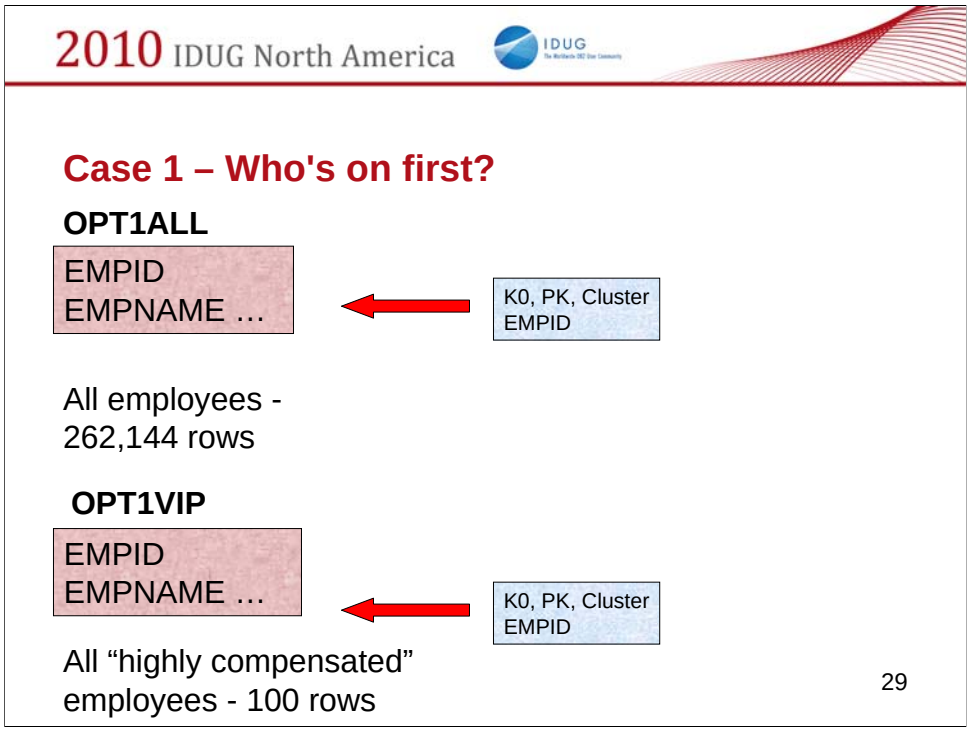


Table structure and stats for Case 1.

Case 1 – Who's on first?

SQL

```
SELECT      *
FROM        OPT1ALL  A
          , OPT1VIP  B
WHERE     A.EMPID    = B.EMPID
```

RUNSTATS BEFORE

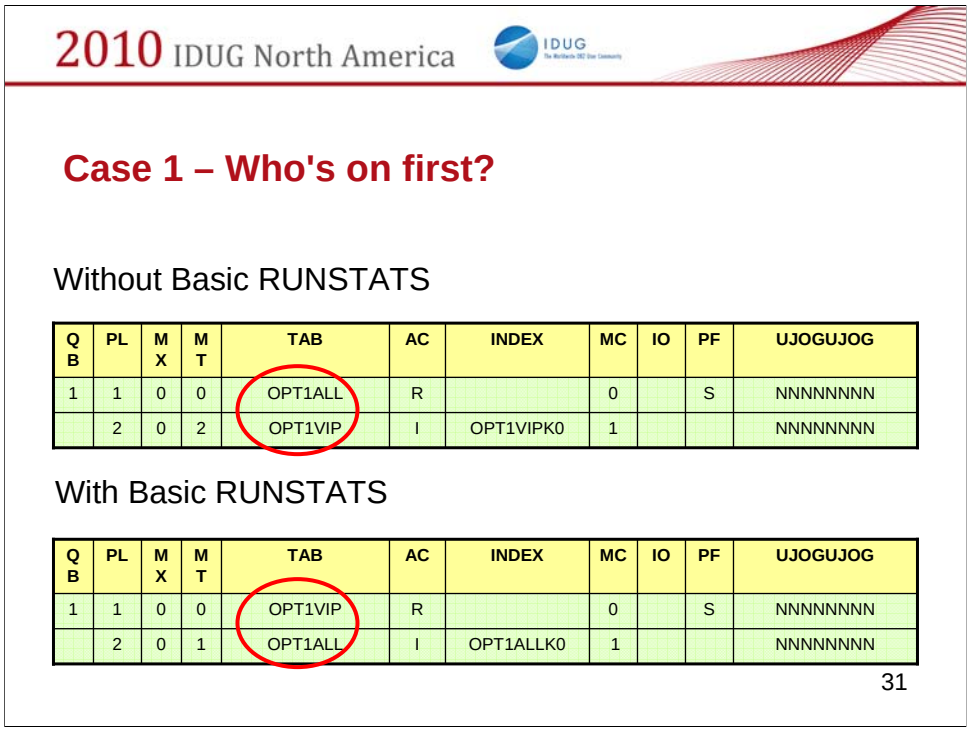
```
None
```

RUNSTATS AFTER

```
RUNSTATS TABLESPACE(..) INDEX(ALL)
```

30

SQL used for testing.



Without basic RUNSTATS:

The bad access path which scans the larger table (262,144 rows) instead of the smaller table (100 rows)

With basic RUNSTATS:

The better access path scanning the smaller table first.

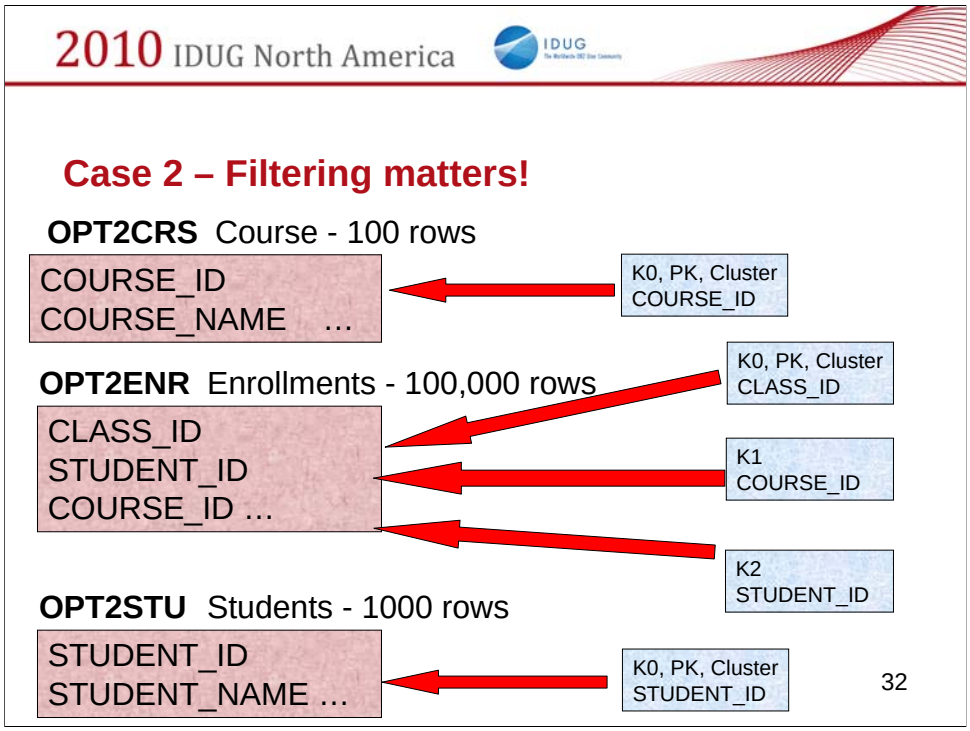


Table structure and stats for Case 2.

Case 2 – Filtering matters!

SQL

```
SELECT    *
FROM      OPT2CRS  C
        , OPT2ENR  E
        , OPT2STU  S
WHERE     C.COURSE_ID       =    E.COURSE_ID
AND       S.STUDENT_ID      =    E.STUDENT_ID
AND       C.COURSE_NAME     LIKE 'DB2%'
AND       S.STUDENT_NAME    LIKE 'S%'
```

RUNSTATS BEFORE

```
None
```

RUNSTATS AFTER

```
RUNSTATS TABLESPACE(..) INDEX(ALL)
```

33

SQL used for testing.

Case 2 – Filtering matters!

Without Basic RUNSTATS

QB	PL	MX	MT	TAB	AC	INDEX	MC	IO	PF	UJOGUJOG
1	1	0	0	OPT2ENR	I	OPT2ENRK1	0			NNNNNNNN
	2	0	4	OPT2CRS	I	OPT2CRSK0	1		L	NNNNNNNN
	3	0	1	OPT2STU	I	OPT2STUK0	1			NNNNNNNN

With Basic RUNSTATS

QB	PL	MX	MT	TAB	AC	INDEX	MC	IO	PF	UJOGUJOG
1	1	0	0	OPT2STU	I	OPT2STUK0	0		S	NNNNNNNN
	2	0	1	OPT2CRS	R		0		S	NNNNNNNN
	3	0	1	OPT2ENR	I	OPT2ENRK2	1			NNNNNNNN

34

Without basic RUNSTATS:

The bad access path using a non-matching index scan against the largest table (OPT2ENR) due to the lack of knowledge about its size.

With basic RUNSTATS:

The better access path which identifies the qualifying students first, the qualifying courses next and then access the largest table using an index.

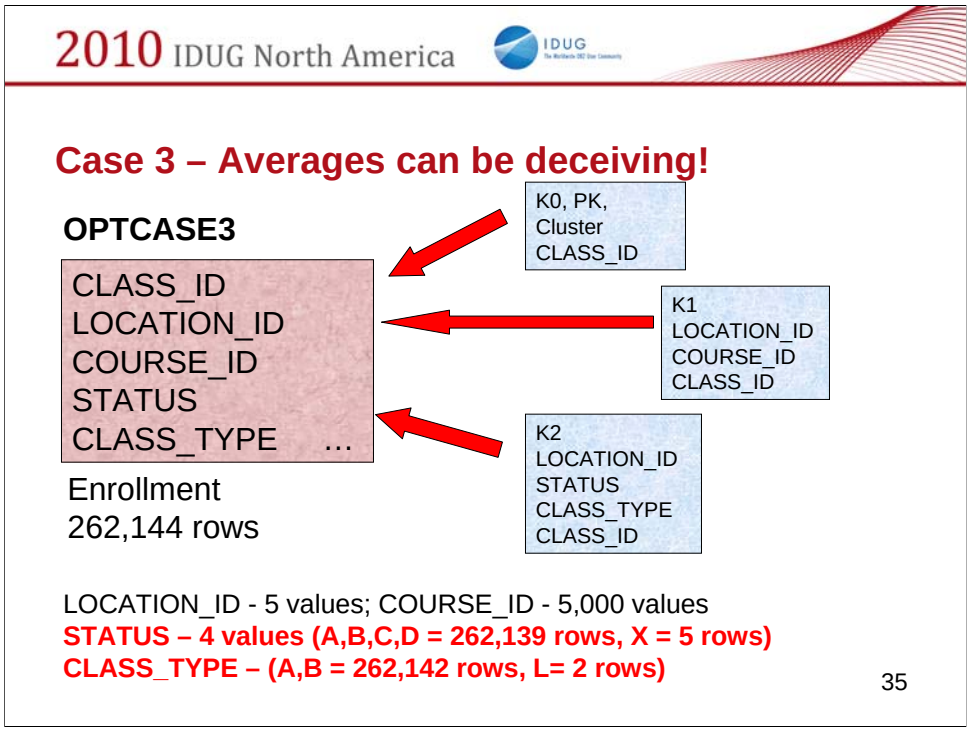


Table structure and stats for Case 3. Note the non-uniform distribution of columns STATUS and CLASS_TYPE.

Case 3 – Averages can be deceiving!

SQL

```
SELECT    *
FROM      OPTCASE3
WHERE     LOCATION_ID    = ?
AND       COURSE_ID      = ?
AND       STATUS         = 'X'
AND       CLASS_TYPE     = 'L'
```

All cancelled classes (status = X) which are labs (class_type = L)

36

SQL used for testing.

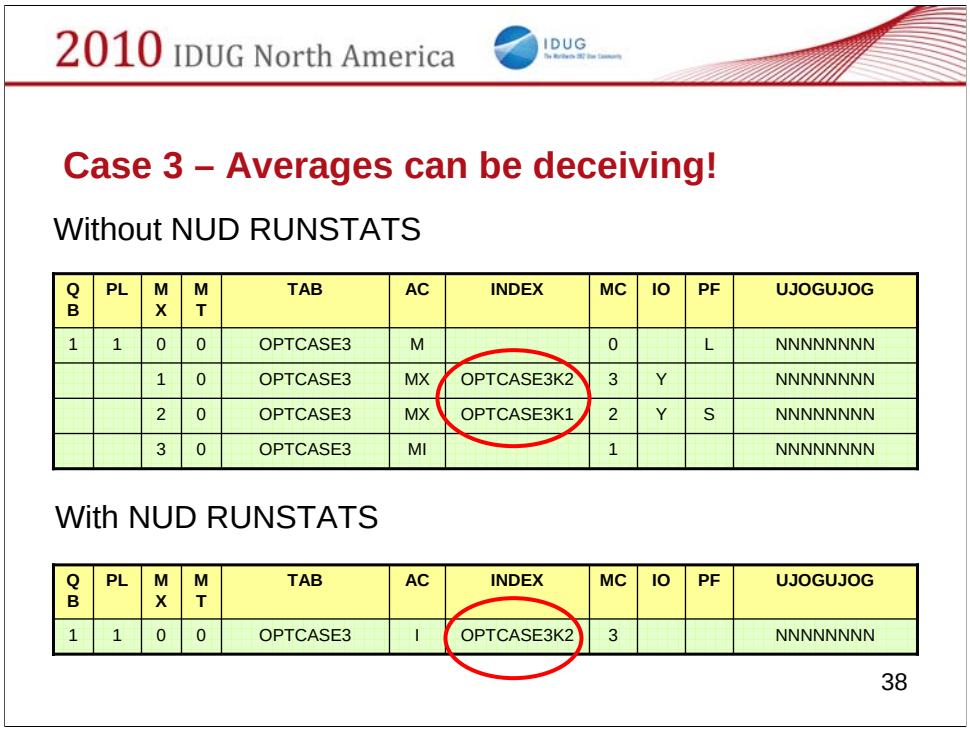
Case 3 – Averages can be deceiving!

RUNSTATS BEFORE

```
RUNSTATS TABLESPACE(..) INDEX(ALL)
```

RUNSTATS AFTER

```
RUNSTATS TABLESPACE(…) TABLE(…)
COLGROUP(STATUS) FREQVAL COUNT 10
COLGROUP(CLASS_TYPE) FREQVAL COUNT 10
```



Without basic RUNSTATS:

The bad access path using a multi-index access with indexes K2 and K1 due to the lack of knowledge about the non-uniform distribution of the data.

With basic RUNSTATS:

The better access path which uses only index K2, which is now known to qualify only a few rows.

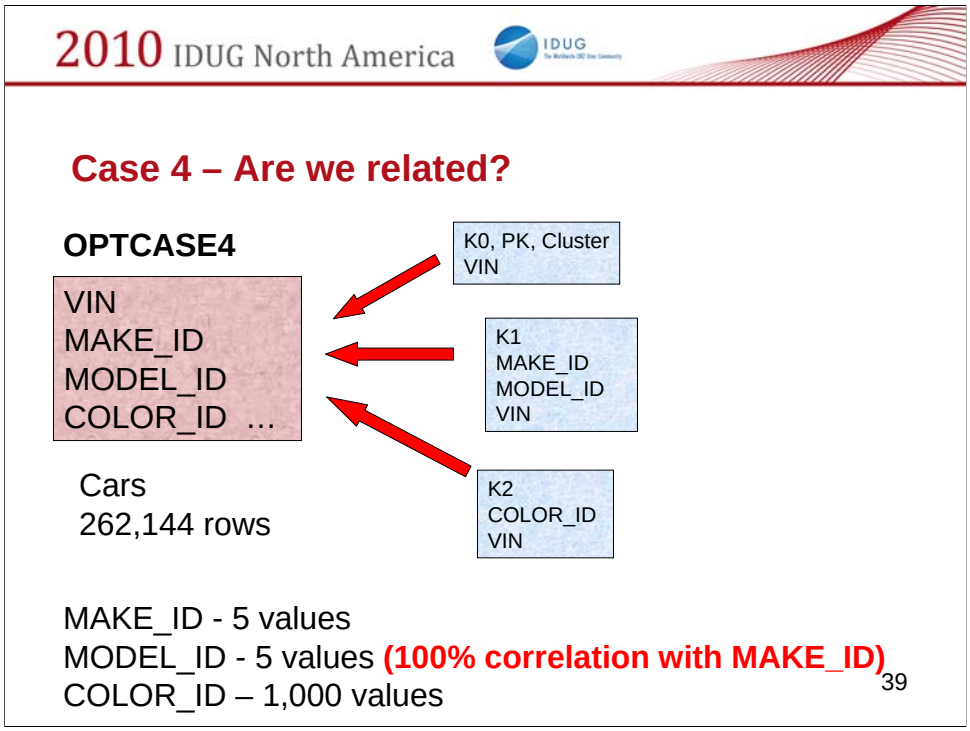


Table structure and stats for Case 4. Note the high correlation between the columns MAKE_ID and MODEL_ID.

Case 4 – Are we related?

SQL

```
SELECT    *
FROM      OPTCASE4
WHERE     MAKE_ID    = ?
AND       MODEL_ID   = ?
AND       COLOR_ID   = ?
```

40

SQL used for testing.

Case 4 – Are we related?

RUNSTATS BEFORE

```
RUNSTATS TABLESPACE(…) TABLE(..)
COLGROUP(MODEL_ID) FREQVAL COUNT 10
COLGROUP(MAKE_ID) FREQVAL COUNT 10
INDEX(ALL)
```

RUNSTATS AFTER

```
RUNSTATS TABLESPACE(…) TABLE(..)
COLGROUP(MODEL_ID) FREQVAL COUNT 10
COLGROUP(MAKE_ID) FREQVAL COUNT 10
INDEX(ALL) KEYCARD FREQVAL NUMCOLS 2
COUNT 10
```

Case 4 – Are we related?

Without CORR RUNSTATS

QB	PL	MX	MT	TAB	AC	INDEX	MC	IO	PF	UJOGUJOG
1	1	0	0	OPTCASE4	M		0		L	NNNNNNNN
		1	0	OPTCASE4	MX	OPTCASE4K2	1	Y		NNNNNNNN
		2	0	OPTCASE4	MX	OPTCASE4K1	2	Y	S	NNNNNNNN
		3	0	OPTCASE4	MI		1			NNNNNNNN

With CORR RUNSTATS

QB	PL	MX	MT	TAB	AC	INDEX	MC	IO	PF	UJOGUJOG
1	1	0	0	OPTCASE4	I	OPTCASE4K2	1		L	NNNNNNNN

42

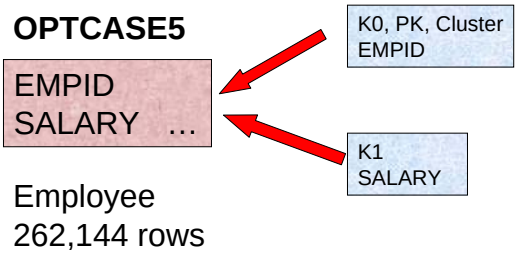
Without basic RUNSTATS:

The bad access path using a multi-index access with indexes K2 and K1 due to the lack of knowledge about the column correlation between MAKE and MODEL.

With basic RUNSTATS:

The better access path which uses only index K2, since K1 provides little filtering.

Case 5 – Ranges are skewed too!



Employee
262,144 rows

Salary range is 20,000 thru 1,000,000
99% of employees fall below 100,000

Table structure and stats for Case 5. Note the range skew.

Case 5 – Ranges are skewed too!

SQL

```
SELECT    *
FROM      OPTCASE5
WHERE     SALARY >= 100,000
```

RUNSTATS BEFORE

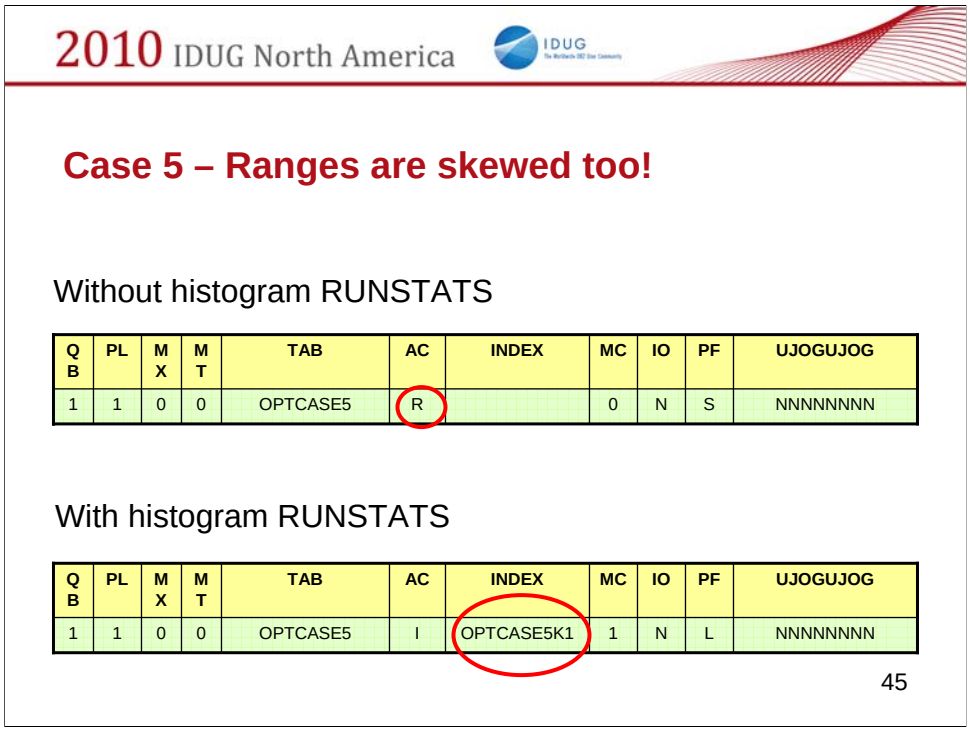
```
RUNSTATS TABLESPACE(..) INDEX(ALL)
```

RUNSTATS AFTER

```
RUNSTATS INDEX(…OPTCASE5K1)
      HISTOGRAM NUMCOLS 1
```

44

SQL used for testing.



Without basic RUNSTATS:

The bad access path, not using K1 because it expects 92% of rows to qualify (1,000,000 – 100,000) / (1,000,000 – 20,000) = 92%

With basic RUNSTATS:

The better access path using K1 because it knows only 3.35 of rows qualify. Note that the calculated filter factor is NOT exact – actual data distribution is such that only 1% lie above 100,000 but depending on where the last quantile starts, DB2 over-estimates this value leading to a conservative estimate of the filtering. In my example, using the default value of 100 quantiles, DB2 chose 76.

In general, such conservative estimates are inevitable due to the "height-balanced" nature of histograms – the sparsely populated intervals tend to be wider as a result.

Case 6 – The "Pick me!" index

- Index and table access review
  - Matching
    - Defines the index slice which must be scanned
    - Based on leading columns of an index only
  - Screening
    - Reduces the number of table touches
    - Table is accessed only if all screening predicates are true (if accessed at all)
    - Based on non-leading columns of the index or any non-indexable columns
  - Result Rows
    - Actual number of rows returned to the application

46

A review of the basic terminology. The stage at which filtering occurs determines what data must be accessed.

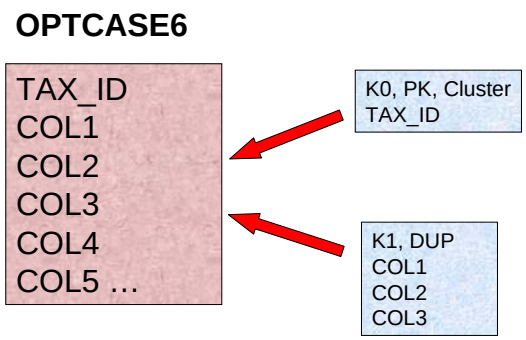
Case 6 – The "Pick me!" index

- What is a "Touch"?
  - A single index row or single table row = 1 Touch
  - Slice of an index = 1st row is random touch, all others are sequential touches
- Response time estimates from QUBE (Quick Upper Bound Estimate)
  - Random read = 10 ms per touch
  - Sequential read (index or table) = 10 ms + 0.01 ms per touch (this accounts for multiple rows per page and sequential prefetch)

47

This case is based on ideas provided in the book on index design by Lahdenmaki and Leach (see (ref #5). We use the estimation formulas from QUBE (Quick Upper Bound Estimate).

Case 6 – The "Pick me!" index



Exercise
262,144 rows

48

Table structure for Case 6.

Note that the detailed calculations for this case cannot possibly covered in the time allotted. So, we will visit them but focus on the conclusions only.

Case 6 – The “Pick me!” index

SQL

```
SELECT    COL5
FROM      OPTCASE6
WHERE     COL1      = ?
AND       COL3      = ?
AND       COL4      = ?
ORDER BY  COL5
```


SQL for testing.

Case 6 – The "Pick me!" index

Table CARD = 1,000,000

Actual individual column cardinalities
CARD(COL1) = 500
CARD(COL3) = 300
CARD(COL4) = 100

Actual combined column cardinalities
CARD(COL1,COL3) = 5,000
CARD(COL1,COL3,COL4) = 250,000

50

Table and column cardinalities.

In general, DB2 will know (via RUNSTATS), CARD(COL1), CARD(COL1,COL2), and CARD(COL1,COL2,COL3) but probably not **CARD(COL1,COL3)**, which is the one that is really needed to determine the correct filter factor.

The optimizer must assume CARD(COL1,COL3) is equal CARD(COL1) × CARD(COL3) = 500 × 300 = 150,000 while the actual CARD(COL1,COL3) is 5000.

Case 6 – The "Pick me!" index

Actual filter factors – average and worst case

Predicate	Average	Worst case (given)
COL1=?	1/500 = 0.2%	2%
COL1=? AND COL3=?	1/5000 = 0.02%	0.5%
COL1=? AND COL3=? AND COL4=?	1/250,000 = 0.0004%	0.005%

51

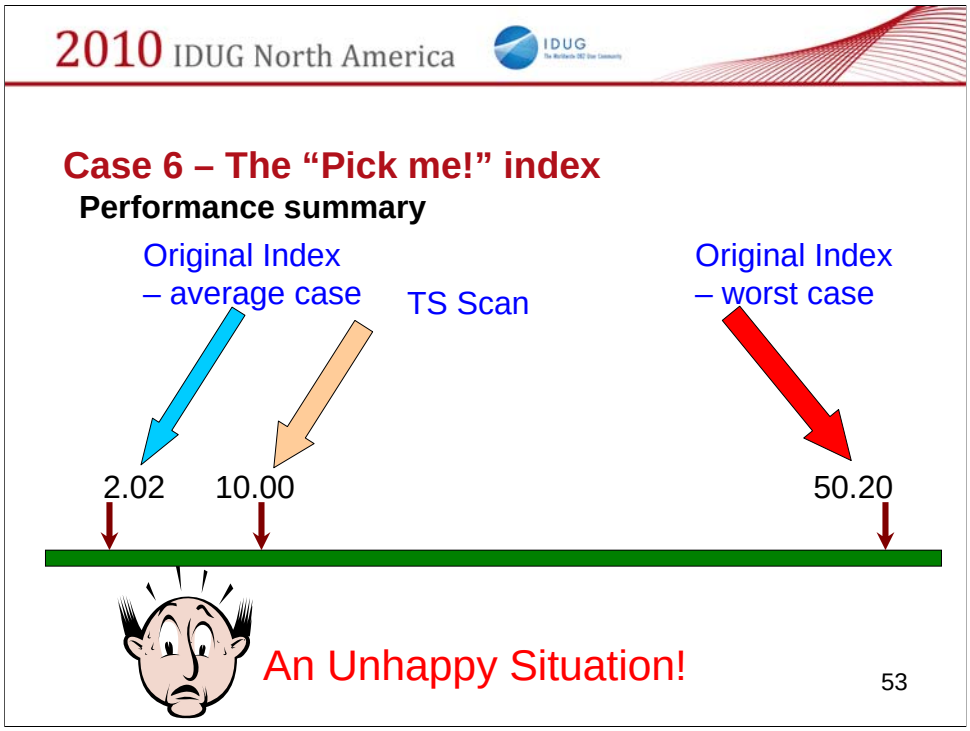
Actual filter factors for the average case and the worst case.

Case 6 – The "Pick me!" index

	TS scan		Index – Avg case		Index – Worst case	
Step	**# Touch**	**Est time**	**# Touch**	**Est time**	**# Touch**	**Est time**
IX			**0. 2%** * 1M = 2,000	2000 * 0.01 ms = 0.020	**2%** * 1M = 20,000	20,000 * 0.01 ms = 0.2
TS	1M	1M * 0.01 ms = 10.00	**0.02%** * 1M = 200	200 * 10 ms = 2	**0.5%** * 1M = 5,000	5,000 * 10 ms = 50
Sort	0.005 * 1M = 50	Small	**0.0004 %** * 1M = 4	Small	**0.005 %** * 1M = 50	Small
Fetch	50	Small & indep.	4	Small & indep.	50	Small & indep.
Total		10.00		2.020		50.20

52

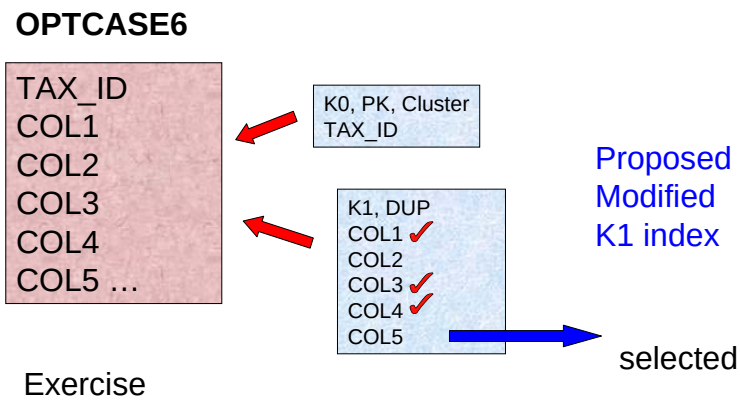
An estimate of the response time with various access path choices.



This poses a dilemma – an access path that is good on average but bad for the worst case!

Also see my discussion of "aggressive vs. defensive optimization" in the next section.

Case 6 – The "Pick me!" index



Creating the proposed "minimal-cost-fat-index". Since this leads to an index-only access, we should fare better.

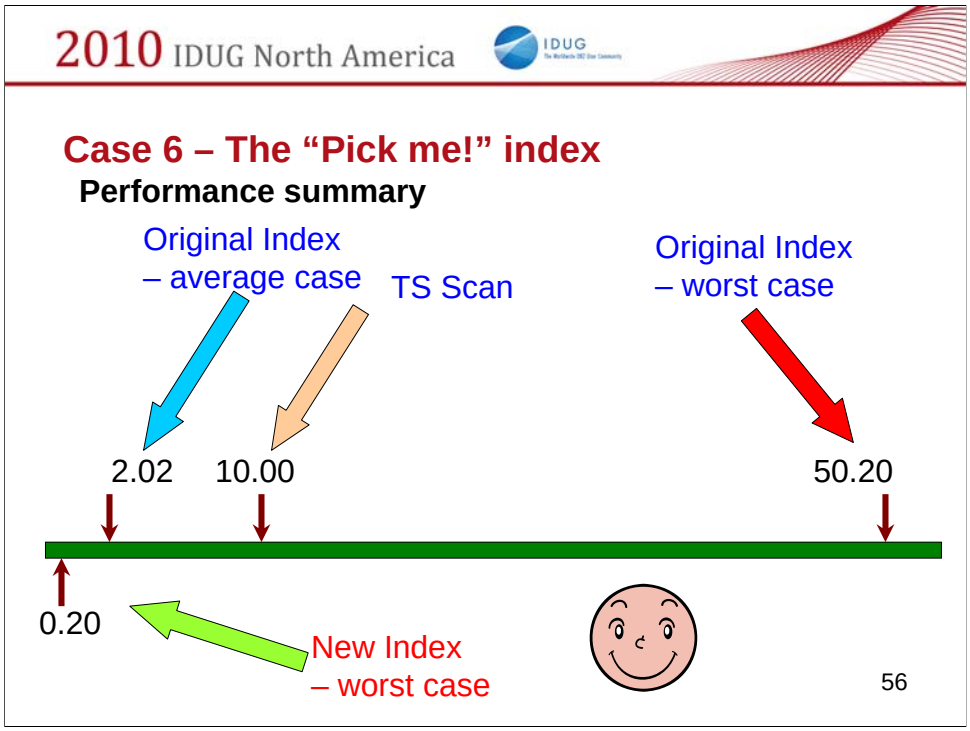
For the query in question an index by columns COL1, COL3, COL4, COL5 would be the best but we are assuming that index consisting of columns COL1, COL2, COL3 is needed and we can only add columns to the end without impacting queries which access by COL1, COL2 or by COL1, COL2, COL3.

Case 6 – The "Pick me!" index

	TS scan		Index – Avg case		Index – Worst case		New Index – Worst case	
Step	**# Touch**	**Est time**	**# Touch**	**Est time**	**# Touch**	**Est time**	**# Touch**	**Est time**
IX	None	0	**0. 2%** * 1M = 2,000	2000 * 0.01 ms = 0.020	**2%** * 1M = 20,000	20,000 * 0.01 ms = 0.2	**2%** * 1M = 20,000	20,000 * 0.01 ms = 0.2
TS	1M	1M * 0.01 ms = 10.00	**0.02%** * 1M = 200	200 * 10 ms = 2	**0.5%** * 1M = 5,000	5,000 * 10 ms = 50	None	0
Sort	0.005 * 1M = 50	Small	**0.0004 %** * 1M = 4	Small	**0.005 %** * 1M = 50	Small	**0.005 %** * 1M = 50	Small
Fetch	50	Small & indep.	4	Small & indep.	50	Small & indep.	50	Small & indep.
Total		10.00		2.020		50.20		0.20

55

An estimate of the response time with an index scan for the worst case – with the new index.



A good access path for all seasons! Life is good!

As explained earlier, real-life situations are far more complex and a minimal-cost-fat-index" may not always be possible or even desirable. However, the example does illustrate how a systematic analysis can lead to a consistently good performance.

It should be noted that we have assumed the Optimizer has full knowledge of the data distribution. In reality, it is likely to know even less making worse choices. The "irresistible" index virtually makes sure it cannot get it wrong!

Where Are We?

1. Cost-based optimizer basics
2. Stats, stats and even more stats
3. Case studies
4. **Access path management - Influencing, stabilizing and falling back**
5. What does the future hold?

This section will cover various methods of influencing and stabilizing the access path.

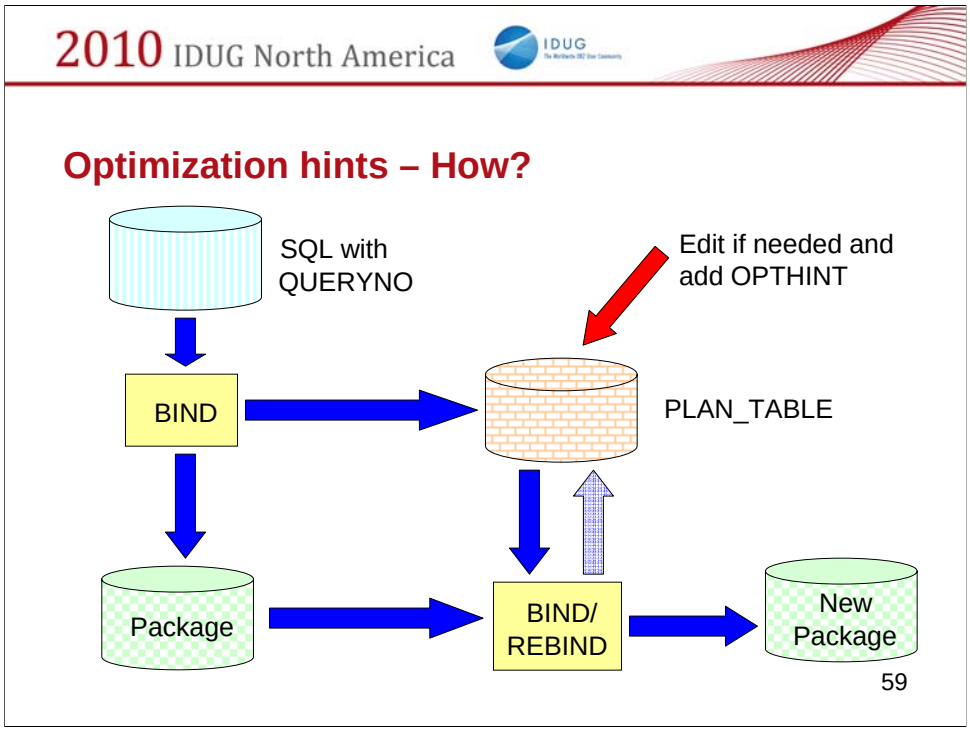
Optimization hints – why?

- Want consistency of response times across rebinds and across code migrations ("**I hate change**") – covered in (a) freezing the access path

- Want to temporarily bypass the access path chosen by DB2 ("**I know better**") – covered in (b) Obtaining a better access path.

58

The business drivers for hints.



While this is the normal mode of operation, you can also construct the plan_table rows from scratch and use them in a bind operation.

Pre-work for Hints

- Specify YES for the DSNZPARM OPTHINTS. If this is not set, all optimization hints are ignored by DB2.
- But… once authorized, cannot limit the use – anyone who can bind the package can apply a hint.
- Before giving hints to DB2, make sure your PLAN_TABLE is of the correct format (see ref #4).
- For best performance, create an ascending index on the following columns of PLAN_TABLE – ***in this order***:
  - QUERYNO
  - APPLNAME
  - PROGNAME
  - VERSION
  - COLLID
  - OPTHINT

60

What is needed to implement hints.

QUERYNO is critical!

- For DB2 hints, the query number clause is optional but..
- If no query number is specified, DB2 uses the statement number.
- Query numbers are critical in the long run, especially for static SQL.
- Dynamic SQL - Statement # is based on application preparing it – e.g. for DSNTEP2/4, same statement # is used.
- Static SQL - In a program with embedded static SQL (e.g. COBOL), **any** program change (e.g. addition of a few comment lines at the top) is likely to affect the statement number and make the hint inapplicable.

61

QUERYNO is critical!!

Simple Hint Examples – Steps

(a) Freezing the access path

	QNO	IX	MC	OPTHINT	HINT_USED
Original →	59	K2	1		
Apply →	59	K2	1	**I_HATE_CHANGE**	
Verify →	59	K2	1		**I_HATE_CHANGE**

(b) Obtaining a better access path

	QNO	IX	MC	OPTHINT	HINT_USED
Original →	59	**K2**	1		
Apply →	59	**K1**	1	**I_KNOW_BETTER**	
Verify →	59	**K1**	1		**I_KNOW_BETTER**

62

What the plan_table looks like before and after.

(a) "Freezing" the access path

1. Determine the query number from the PLAN_TABLE. Using QUERYNO in the SQL helps correlate the query number with the query in the application.
2. Make the PLAN_TABLE rows for that query into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is **I_HATE_CHANGE**.
3. Tell DB2 to use the hint, and verify in the PLAN_TABLE that DB2 used the hint. The steps depend on whether you have dynamic SQL or static SQL (see next 2 slides).

63

For "freezing" a good access path.

(a) "Freezing" the access path

3a. **Dynamic SQL**

- Execute the SET CURRENT OPTIMIZATION HINT statement in the program to tell DB2 to use **I_HATE_CHANGE**. For example: SET CURRENT OPTIMIZATION HINT = '**I_HATE_CHANGE**'.
- If you do not explicitly set the CURRENT OPTIMIZATION HINT special register, the value that you specify for the bind option OPTHINT is used – in that case, rebind the package to pick up the SET CURRENT OPTIMIZATION HINT statement.

(a) "Freezing" the access path

3a. **Dynamic SQL (contd.)**

- Execute the EXPLAIN statement on the SQL statements for which you requested DB2 to use **I_HATE_CHANGE**. This step adds rows to the PLAN_TABLE for those statements. The rows contain **I_HATE_CHANGE** in the HINT_USED column.
- If DB2 uses all of the hints that you provided, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the hints.
- If any of your hints are invalid or not used, DB2 issues SQLCODE +395.

65

If DB2 issues SQLCODE +395 a reason code is also returned. About 40 reason codes are possible (e.g. **15 =** Specified index cannot be used as requested, **19 =** Nested loop join cannot be done as requested etc.). This is very useful in diagnosing what the actual error is.

(a) "Freezing" the access path

3b. **Static SQL**

- Rebind the package that contains the statements. Specify bind options EXPLAIN(YES) and OPTHINT('**I_HATE_CHANGE**') to add rows for those statements in the PLAN_TABLE that contain **I_HATE_CHANGE** in the HINT_USED column. If DB2 uses the hint you provided, it returns SQLCODE +394 from the rebind. If your hints are invalid or not used, DB2 issues SQLCODE +395.

66

If DB2 issues SQLCODE +395 a reason code is also returned. About 40 reason codes are possible (e.g. **15 =** Specified index cannot be used as requested, **19 =** Nested loop join cannot be done as requested etc.). This is very useful in diagnosing what the actual error is.

(a) "Freezing" the access path

4. Select from PLAN_TABLE to see what was used for the last rebind. It should show the **I_HATE_CHANGE** hint, as the value in OPTHINT and it should also show that DB2 used that hint, indicated by **I_HATE_CHANGE** in the HINT_USED column (not on the original row).

	QNO	IX	MC	OPTHINT	HINT_USED
Original →	59	K2	1		
Apply →	59	K2	1	**I_HATE_CHANGE**	
Verify →	59	K2	1		**I_HATE_CHANGE**

67

(b) Obtaining a better access path

1. Put the old access path in the PLAN_TABLE and associate it with a query number.
2. Make the PLAN_TABLE rows into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is **I_KNOW_BETTER**.
3. Change the access path so that PLAN_TABLE reflects the rows associated with the desirable new access path **I_KNOW_BETTER**.
4. Tell DB2 to look for the hint for this query:
   - If dynamic, issue: SET CURRENT OPTIMIZATION HINT = '**I_KNOW_BETTER**';
   - If static, rebind the package or plan with OPTHINT set to **I_KNOW_BETTER**.

68

… and for obtaining a better one.

(b) Obtaining a better access path

5. Use EXPLAIN on the query or package to check the results.
6. Check the PLAN_TABLE which should show **I_KNOW_BETTER** as the OPTHINT and it should also show that DB2 used that hint with **I_KNOW_BETTER** as the HINT_USED (not on the original row).

	QNO	IX	MC	OPTHINT	HINT_USED
Original ➡	59	**K2**	1		
Apply ➡	59	**K1**	1	**I_KNOW_BETTER**	
Verify ➡	59	**K1**	1		**I_KNOW_BETTER**

69

Locating the hint

- For a PLAN_TABLE row, the QUERYNO, APPLNAME, PROGNAME, VERSION, and COLLID values must match the corresponding values for an SQL statement
- In addition:
  - If the SQL statement is executed dynamically, the OPTHINT value for that row must match the value in the CURRENT OPTIMIZATION HINT special register.
  - If the SQL statement is executed statically, the OPTHINT value for the row must match the value of bind option OPTHINT for the package or plan that contains the SQL statement.

70

How does DB2 locate the hint in the plan_table?

Validating the hint

- DB2 validates only the following PLAN_TABLE columns:
  - METHOD
  - CREATOR and TNAME
  - TABNO
  - ACCESSTYPE
  - ACCESSCREATOR and ACCESSNAME
  - SORTN_JOIN and SORTC_JOIN
  - PREFETCH
  - PAGE_RANGE
  - PARALLELISM_MODE
  - ACCESS_DEGREE and JOIN_DEGREE
  - WHEN_OPTIMIZE
  - PRIMARY_ACCESSTYPE
- If the access path you suggest cannot be enforced, all hints of that QBLOCK are discarded.

71

Rules for validating the hint.

Hint limitations

- Hints cannot force or undo query transformations, such as subquery transformation to join or materialization or merge of a view or table expression.
- If a query is not transformed in that release, but in a later release of DB2 it is transformed, DB2 does not use the hint in the later release.
- Be aware that a hint supplied on the PLAN_TABLE which was ignored (did not match the OPTHINT specified) is also shown but results in return code 0 for the BIND – must check for +394 not 0!

```
DSNT222I  :DBxx DSNTBBP2 REBIND WARNING
FOR PACKAGE = DBxx.xxxxxxx.HINTNEW.
USE OF OPTHINT RESULTS IN
1 STATEMENTS WHERE OPTHINT FULLY APPLIED
0 STATEMENTS WHERE OPTHINT NOT APPLIED OR PARTIALLY APPLIED
1 STATEMENTS WHERE OPTHINT IS NOT FOUND
```

Some of the limitations.

Catalog stats manipulation

- If you update stats, make sure you update **all** related stats – e.g. if the table card is increased, the stats for the associated indexes and column should be updated also.
- Can help your specific query, but other queries can be affected adversely.
- The UPDATE statements must be repeated after RUNSTATS resets the catalog values.
- If you are using dynamic statement caching, you must invalidate statements in the cache that access those tables or indexes.
  - For this, you can use:

```
RUNSTATS ..REPORT NO UPDATE NONE
```

73

Manipulating the catalog stats to influence the access path.

SQL “tricks”

- Some of the examples of such “tricks” are:
  - Add predicates to turn indexable predicates into stage 2 (bad) e.g. OR 0=1
  - Add predicates to turn indexable predicates into stage 1 non-indexable (good) e.g. col +0
  - Add fake redundant predicates to favor one access path over another
  - Add OPTIMIZE FOR n ROWS where n is artificially small or large
  - Define a table as VOLATILE to encourage index usage
  - Use CARDINALITY or CARDINALITY MULTIPLIER clause of a user-defined function
- “Tricks” can cause significant performance degradation if they are not carefully implemented and monitored. In case of a query re-write, the “trick” may become ineffective in a future release of DB2.

74

Using “tricks” to influence the access path.

Package Stability - Framework

- Ability to easily fallback to a previous (or original) copy of a version of a package (Note: Each version can have up to 3 copies, version is **NOT** the same as copy!)
- The support applies to packages — not plans — and includes non-native SQL procedures, external procedures, and trigger packages.
- Some IBM manuals refer to this feature as “Plan stability” since it deals with the stability of access paths (an “access plan”).
- I prefer the term package stability since this option is available for packages (not plans).

75

The infrastructure needed.

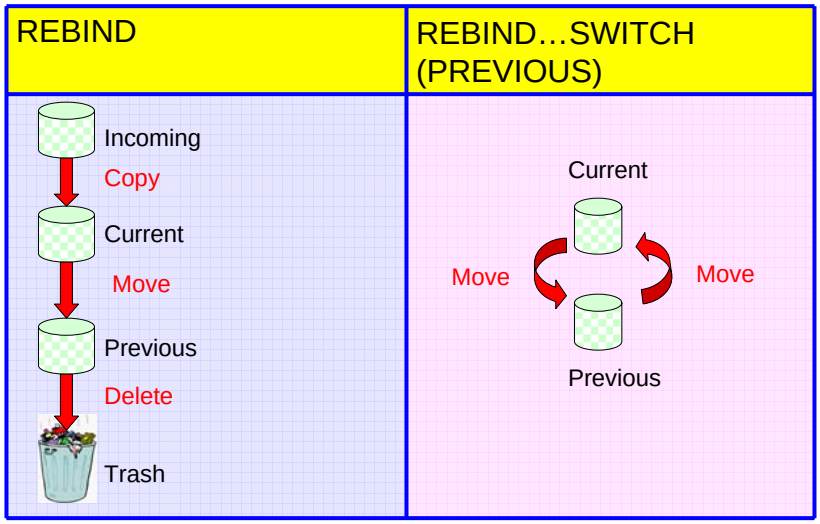
Controlling package stability

- The option can be controlled at two levels:
  - Subsystem level via a new DSNZPARM PLANMGMT (suggest not setting this to use BASIC or EXTENDED!)
  - BIND level with new options for REBIND
- Possible settings:
  - PLANMGMT(OFF) – 1 copy
  - PLANMGMT(BASIC) – 2 copies
  - PLANMGMT(EXTENDED) – 3 copies

76

How do you control it (and at what level)?

PLANMGMT(BASIC) – REBIND and SWITCH



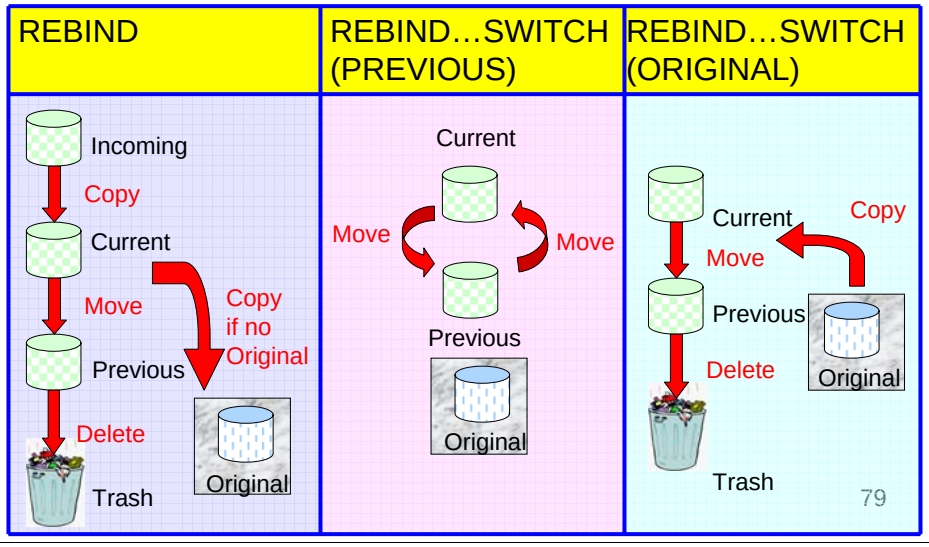
77

How does it work?

PLANMGMT(BASIC)

- The package has one active ("current") copy, and one additional ("previous") copy is preserved.
- At each REBIND:
  - Any previous copy is discarded
  - The current copy becomes the previous copy
  - The incoming copy becomes the current copy
- If you issue two or more rebinds after migration to a new version, you will wipe out the access path for packages from previous version which you might want to preserve.

PLANMGMT(EXTENDED) – REBIND and SWITCH



PLANMGMT(EXTENDED)

- Retains up to three copies of a package: one active copy and two additional old copies (PREVIOUS and ORIGINAL) are preserved.
- At each REBIND:
  - Any previous copy is discarded
  - If there is no original copy, the current copy is saved as the original copy
  - The current copy becomes the previous copy
  - The incoming copy becomes the current copy
- Unlike the case when you use PLANMGMT(BASIC), the original copy is the one that existed from the "beginning", **it is saved once and never overwritten** (it could be the copy you wish to preserve from a prior version).

Package Stability – Things to note

- During REBIND PACKAGE with active PLANMGMT, the various copies of the package are managed in the DB2 directory.
- Extra rows in SYSPACKDEP, denoted by different values for DTYPE, indicate the availability of additional package copies.
- DB2 stores all of the details about a package in the Directory, but only the **active** copy is externalized in SYSPACKAGE (SYSPACKDEP does contain it).
- If a dependent object is dropped that causes invalidation of the original or previous copy of the package, this is not visible in the Catalog until a REBIND with the SWITCH option activates that copy of the package.

81

of copies for a package

```
SELECT      SP.COLLID, SP.NAME, SP.VERSION,
            COUNT(DISTINCT SPD.DTYPE)
                                AS COPY_COUNT
FROM        SYSIBM.SYSPACKAGE   SP
          , SYSIBM.SYSPACKDEP   SPD
WHERE       SP.COLLID  =        SPD.DCOLLID
AND         SP.NAME    =        SPD.DNAME
GROUP BY    SP.COLLID, SP.NAME, SP.VERSION
```

- COPY_COUNT=1: OFF (DTYPE = blank)
- COPY_COUNT=2: BASIC (DTYPE = blank and P)
- COPY_COUNT=3: EXTENDED (DTYPE = blank, P and O)

82

How do you tell which flavor of package stability (if any) applies to a package?

Deleting old copies

- A new FREE PACKAGE option called PLANMGMT SCOPE allows you to free older copies that are no longer necessary.
  - PLANMGMT SCOPE(ALL) - To free the entire package including all copies. This is the default.
  - PLANMGMT SCOPE(INACTIVE) - To free all old copies only (i.e. original and previous, if any).
- The existing FREE PACKAGE command and DROP TRIGGER SQL statement drops the specified package and trigger as well as all associated current, previous and original copies – i.e. it behaves like SCOPE(ALL).

83

Clean up of old copies.

Influencing the access path - Summary

Criteria	Hints	Stats Plug	SQL Tricks	Package Stability
Ease	External but need to manage	Cumbersome	Need to re-compile	Fallback only
Granularity	SQL	All SQL	SQL	Package
Performance impact	None	Could hurt other SQL	May hurt if stage 2	N/A
Future impact	Generally permanent (but need to manage QUERYNO)	Generally permanent	May regress	N/A
DASD/EDM Pool requirements	N/A	N/A	N/A	Double or triple SPT01 (use compression)

84

Comparing the options. The best option depends entirely on your setup.

Another option worth mentioning is the ability to bind into a dummy collection. A more detailed discussion on managing access path appears in my other session (A15).

Where Are We?

1. Cost-based optimizer basics
2. Stats, stats and even more stats
3. Case studies
4. Access path management - Influencing, stabilizing and falling back
5. What does the future hold?

In this final section, we will present what the future holds (announced) and what I would like to hold.

V10 Announced Changes

- Safe optimization
  - Choose an access path with a slightly higher cost estimate if it has a higher cost certainty - earlier, possible to overestimate the filtering of a predicate (range predicate, NUD or with host variables or parameter markers).
- Improved list prefetch processing (no RID pool failures).
- Index matching on multiple IN-list predicates.
- Predicate transitive closure for IN-list predicates.
- Access path repository and ability to switch – even for BIND
- Hints at the statement level
- SQL Pagination

86

Some of the many enhancements which will be covered in other IBM sessions. I can mention only those features which have been announced by IBM.

Safe optimization is a great start! But also see my discussion of defensive optimization.

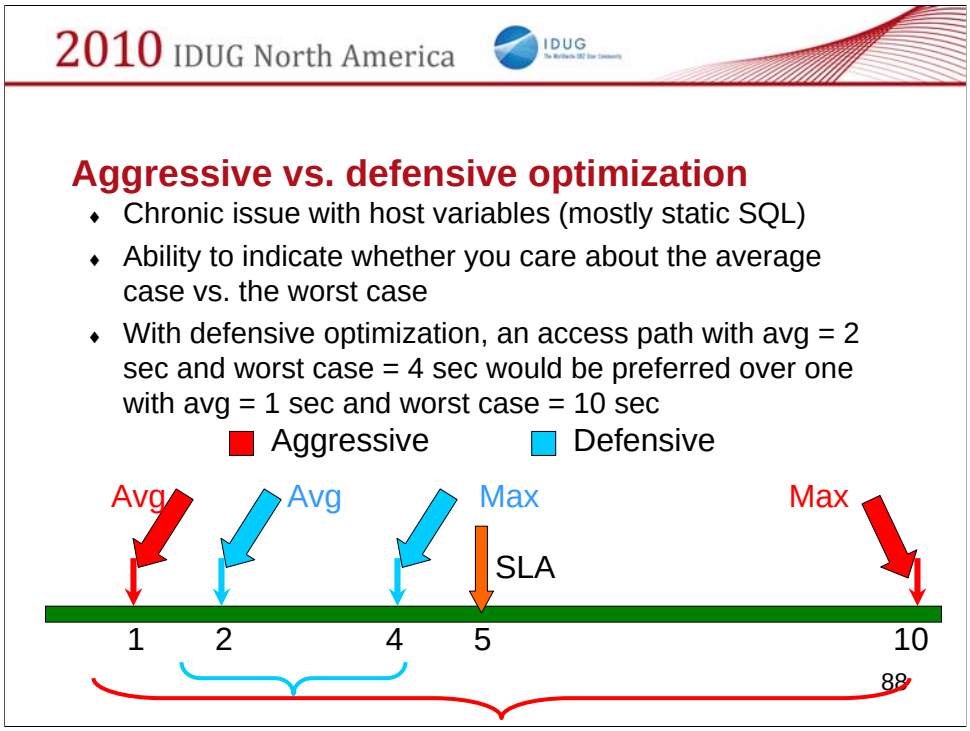
The ability to retain multiple copies of access paths extends what Package Stability provided much further – I am excited about this one!

My wish list (Are you listening, IBM?)

- Aggressive vs. defensive optimization
- Scrolling/Re-positioning issues
- Externalization of alternatives and reasons
- “Anti-hints”

87

A list of enhancements I would like to see some day… we will discuss each one in detail in the following slides.



In several cases, the average is not what matters but the spikes do – with financial penalties associated with them. This allows you to pay a higher premium to flatten out the spikes.

While DB2 10 Safe Optimization (discussed earlier) considers uncertainty, I would like it to be more granular statement level. This would place the control (and the burden!) on the DBA/developer.

Scrolling/Re-positioning issues

- Most scrolling predicates provide no filtering at all and easily mislead the optimizer – example:

```
WHERE SALARY > :WS-SALARY
                ====Vs.=====
WHERE FUND > :WS-FUND
                OR
          (FUND =  :WS-FUND AND
           ACCT >  :WS-ACCT)
                OR
           (FUND =  :WS-FUND AND
           ACCT =  :WS-ACCT AND
           DATE >  :WS-DATE)
```

- A special SQL clause like joining – “SCROLL ON” to recognize such constructs

89

Range predicates used in scrolling logic are notoriously hard for the optimizer and the filter factors tend to be too aggressive The optimizer really has no way know if proper filtering is going to occur without a special indication which indicates scrolling.

Externalization of alternatives and reasons

- An extended EXPLAIN command to show not only the access path chosen but also the alternatives considered and the reasons they were discarded.
- Externalization of i/o cost, cpu cost etc.

90

Finally, making the optimizer translucent (if not transparent). This would also help in picking the alternative when using OPTHINTs, which is not always clear.

“Anti-hints”

- Knowing the desirable access path is not always obvious
- Creating a desirable access path can be cumbersome
- This would allow you to say “not this, try something else” and let the optimizer pick an alternative

The lazy man’s alternative.

Summary

1. Cost-based optimizer basics
   - Filtering, Effect of NUD, Effect of correlation, Effect of clustering, Literals and REOPT
2. Stats, stats and even more stats
   - Basic stats, Distribution stats, Correlation stats, Histogram
3. Case studies – searching for the “irresistible” index
   - Table sizes (basic stats), Table filters (basic stats), Non uniform distribution, correlation, range skew (Histogram stats), The "Pick me!" index
4. Influencing the access path
   - Stats plugging, Tricks, Hints, Package stability
5. What does the future hold?
   - V10 announced changes, My wish list

92

I trust this session has empowered you with the knowledge to exploit the optimizer fully. Good Luck!

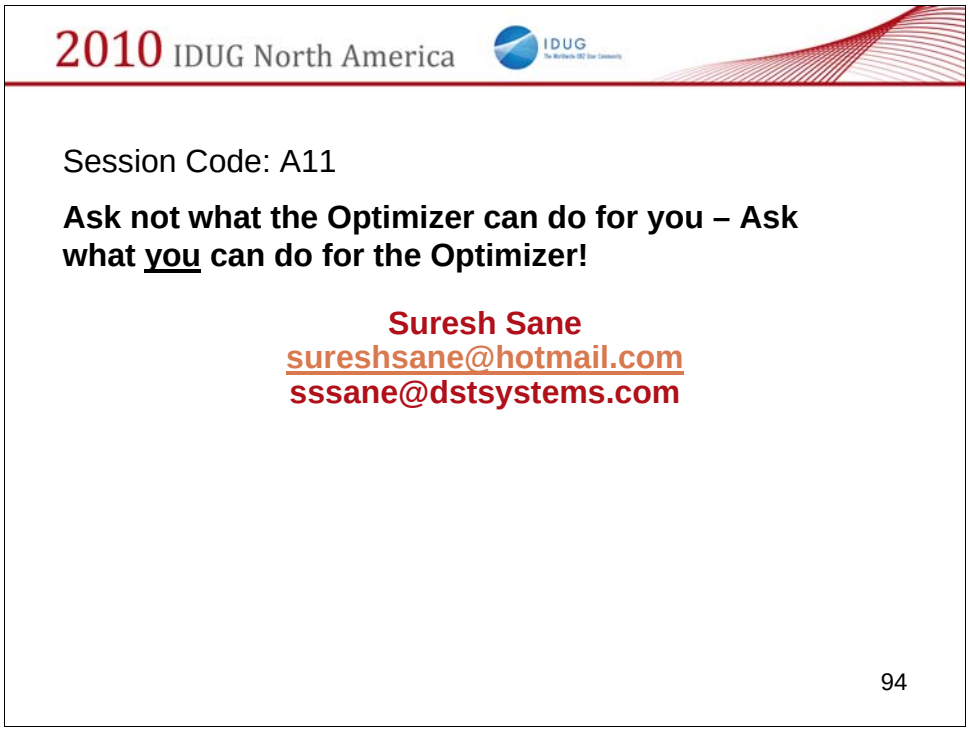
References

1. IBM Redbook – "DB2 9 for z/OS – Packages revisited - SG24-7688"
2. IB Redbook – "DB2 9 for z/OS - Technical Overview - SG24-7330"
3. IBM Redbook – "DB2 9 for z/OS - Performance Topics - SG24-7473"
4. DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide - SC18-9851
5. "Relational Database Index Design and the Optimizers" - Tapio Lahdenmaki and Mike Leach
6. IBM Redbook – "DB2 - Quick Upper Bound Estimate An Application Design Methodology - SG24-2549"

Some of the many useful references. While reference #6 is now dated, the ideas presented in it are still useful for a quick analysis.

ACKNOWLEDGEMENTS:

I would like to express my sincere thanks to Adarsh Pannu (IBM Optimizer team) for his review of a draft copy and suggestions for improvement. Thank you Adarsh!



Thank you and good luck with access paths… the Optimizer is not perfect, but it is your friend – treat it like one!