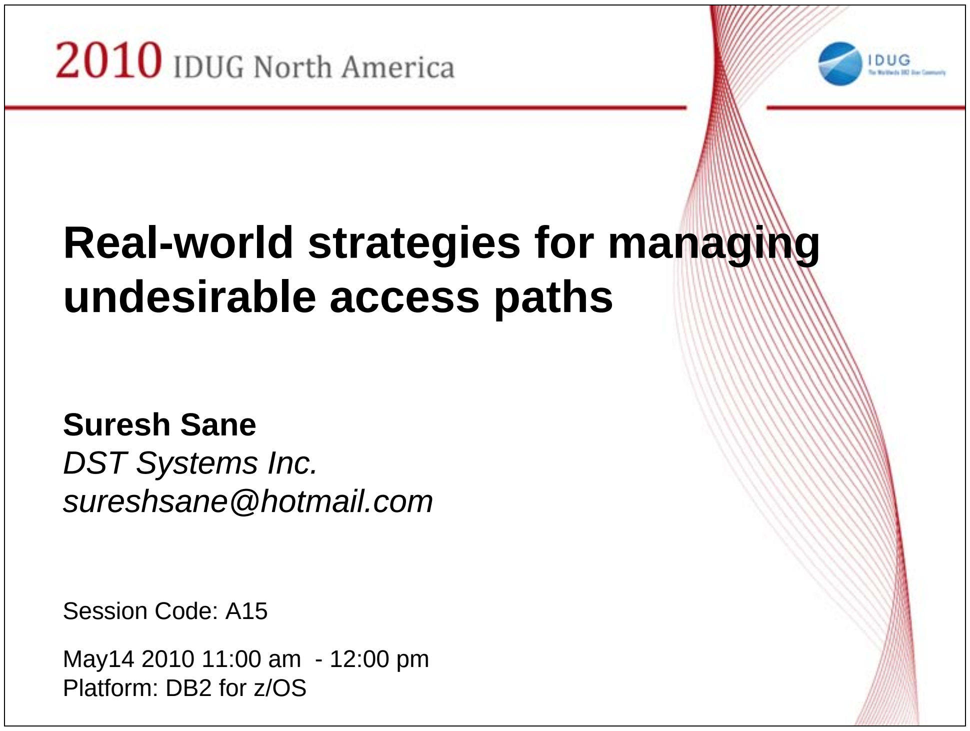


2010 IDUG North America

Real-world strategies for managing undesirable access paths

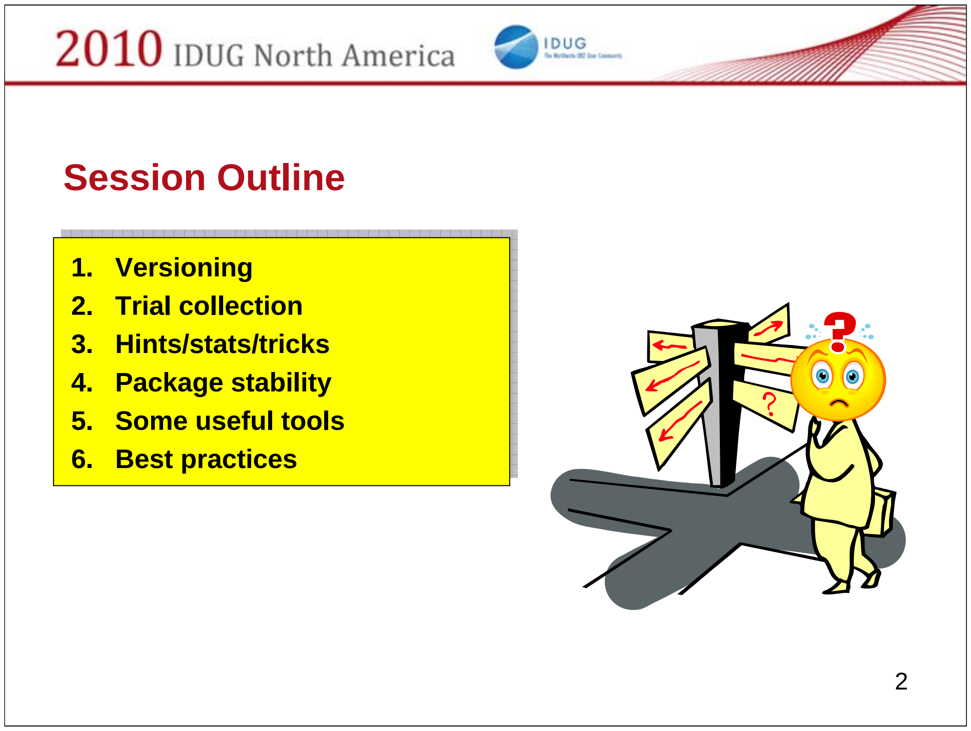
Suresh Sane
DST Systems Inc.
sureshsane@hotmail.com

Session Code: A15

May14 2010 11:00 am - 12:00 pm
Platform: DB2 for z/OS

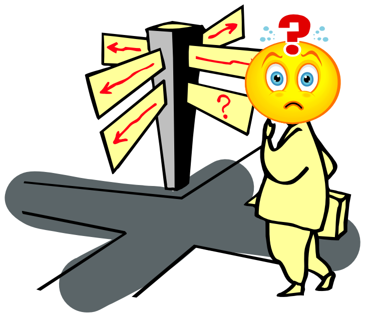
In a DB2 production environment, almost all software changes are typically tightly controlled, but the impact of production binds and rebinds is not. This can result in unpredictable performance since it must be dealt with in a reactive mode. In this session, we focus on strategies to minimize the surprises and to fallback gracefully if hit with an undesirable change. This includes a discussion of the Access Path Stability feature available in DB2 9.

We will mention some tools, but the focus is on the design choices and the strategic planning.

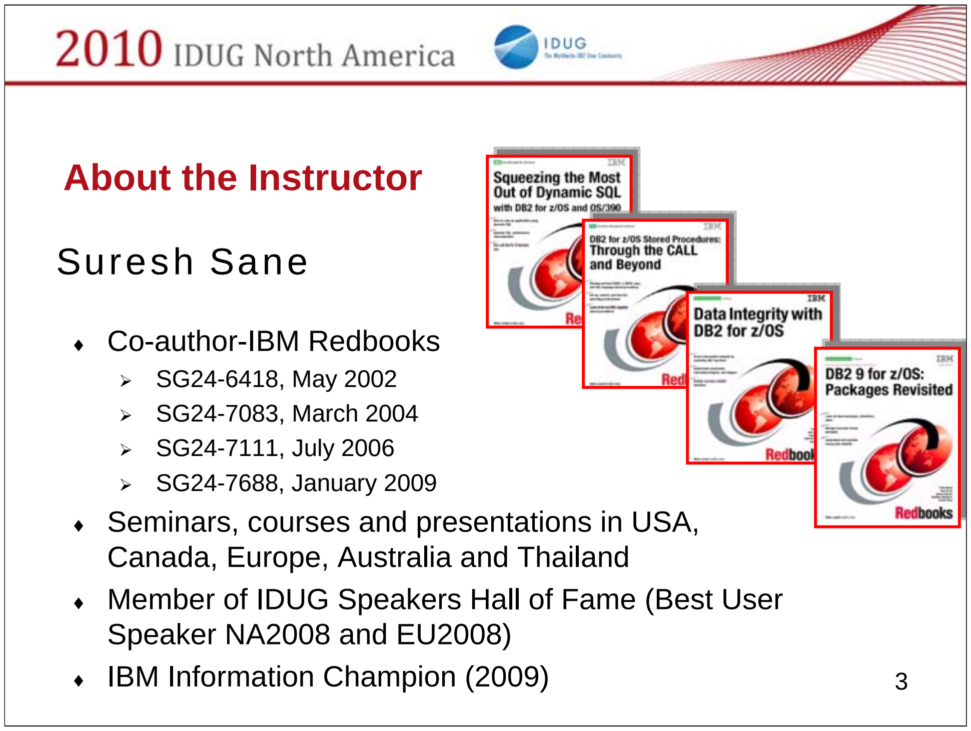


Session Outline

1. **Versioning**
2. **Trial collection**
3. **Hints/stats/tricks**
4. **Package stability**
5. **Some useful tools**
6. **Best practices**



•Versioning
  •Infrastructure
  •Migration
  •Fallback
•Trial collection
  •Why trial?
  •Trail directly in prod
  •Trial with DEFINE(NO)
  •Trial with program promotion
•Hints/stats/tricks
  •Hints
  •Stats
  •Tricks
  •Advantages and disadvantages
•Package stability
  •Infrastructure
  •Description
  •Advantages and disadvantages
•Some useful tools
  •IBM Bind Manager
  •IBM Path Checker
  •BMC SQL Explorer for DB2
  •Neon Bind Impact Expert
  •EZ- Impact Analyzer for DB2 on z/OS
•Best practices
  •Comparison
  •Summary
  •Recommendations



2010 IDUG North America

About the Instructor

Suresh Sane

- Co-author-IBM Redbooks
  - SG24-6418, May 2002
  - SG24-7083, March 2004
  - SG24-7111, July 2006
  - SG24-7688, January 2009
- Seminars, courses and presentations in USA, Canada, Europe, Australia and Thailand
- Member of IDUG Speakers Hall of Fame (Best User Speaker NA2008 and EU2008)
- IBM Information Champion (2009)

3

Squeezing the Most Out of Dynamic SQL with DB2 for z/OS and OS/390
DB2 for z/OS Stored Procedures: Through the CALL and Beyond
Data Integrity with DB2 for z/OS
DB2 9 for z/OS: Packages Revisited
Redbooks

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

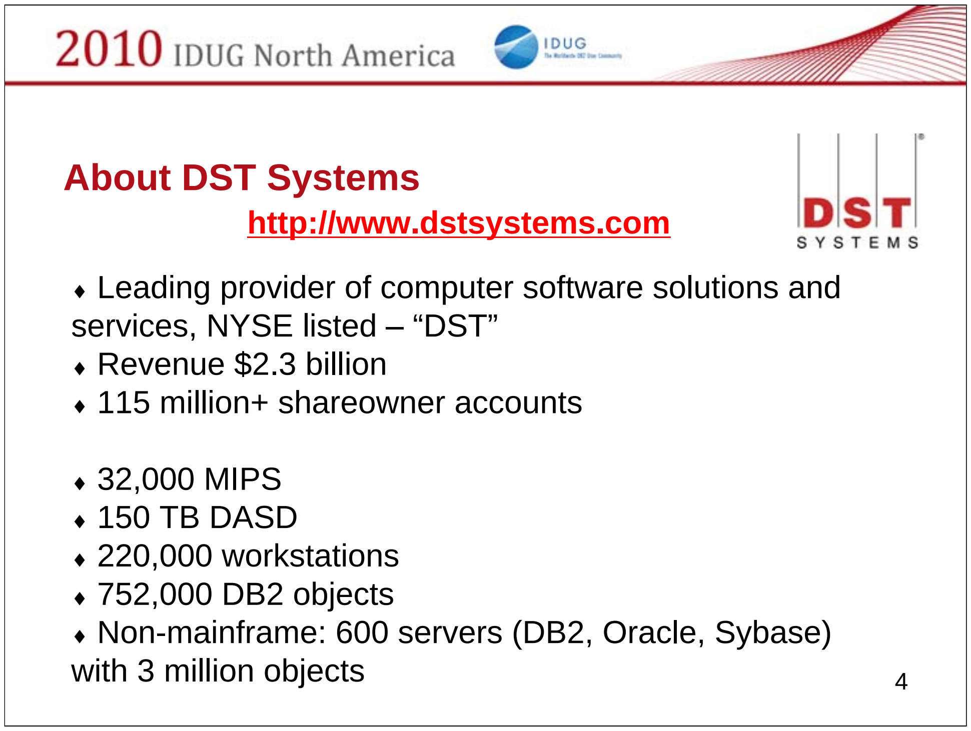
DST Systems, Inc.

1055 Broadway

Kansas City, MO 64105

USA

(816) 435-3803


2010 IDUG North America

About DST Systems

http://www.dstsystems.com

- Leading provider of computer software solutions and services, NYSE listed – “DST”
- Revenue $2.3 billion
- 115 million+ shareowner accounts

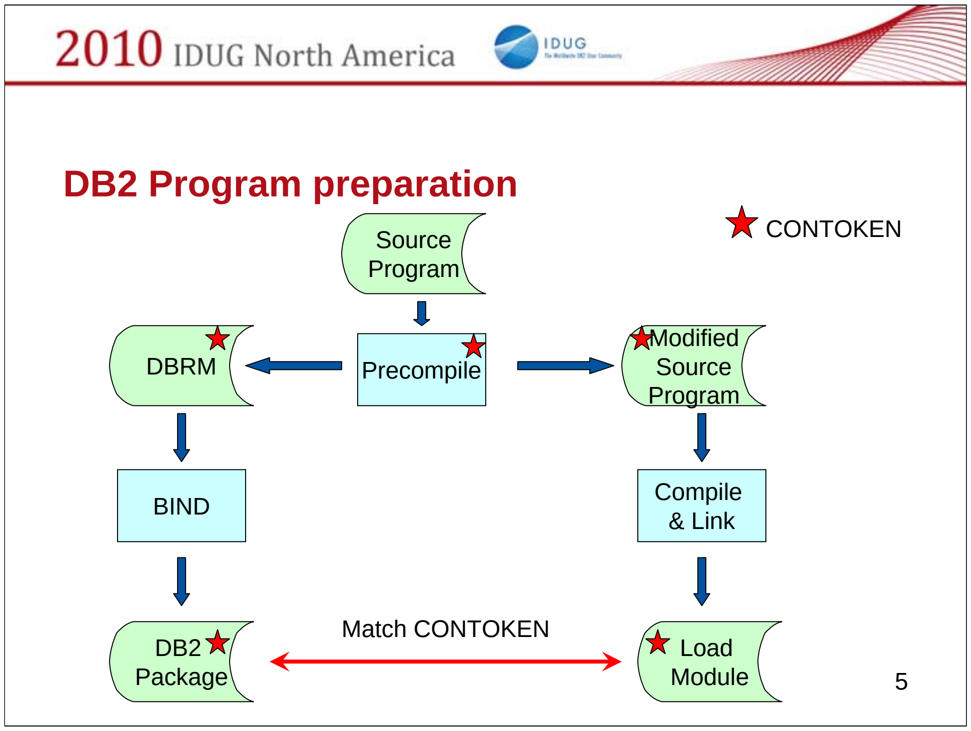
- 32,000 MIPS
- 150 TB DASD
- 220,000 workstations
- 752,000 DB2 objects
- Non-mainframe: 600 servers (DB2, Oracle, Sybase) with 3 million objects

4

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.

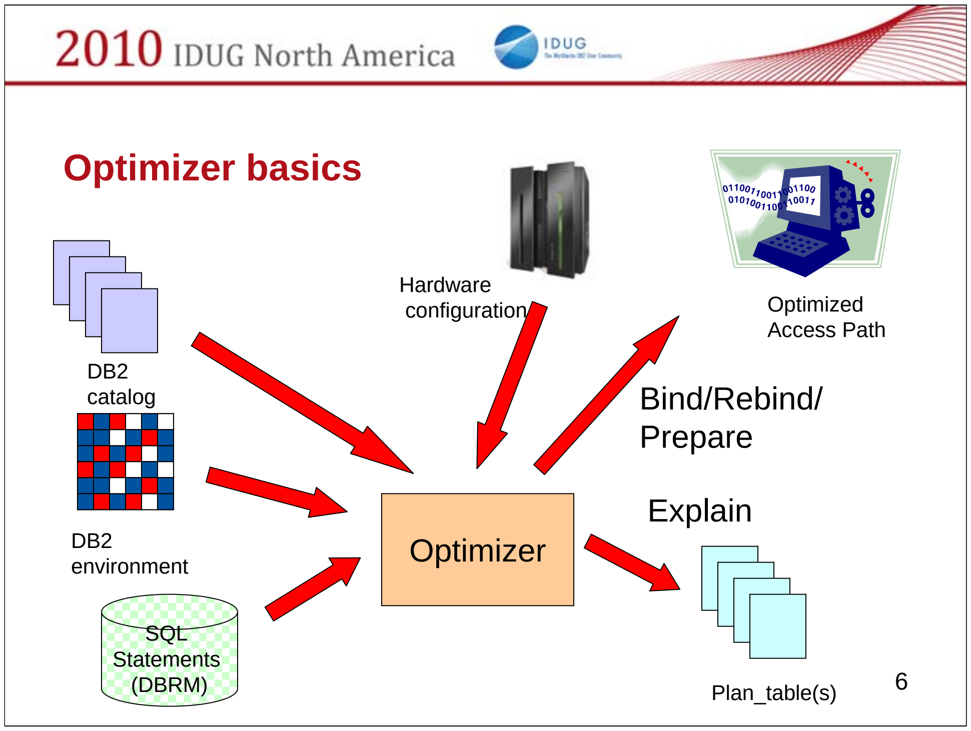


2010 IDUG North America

DB2 Program preparation

5

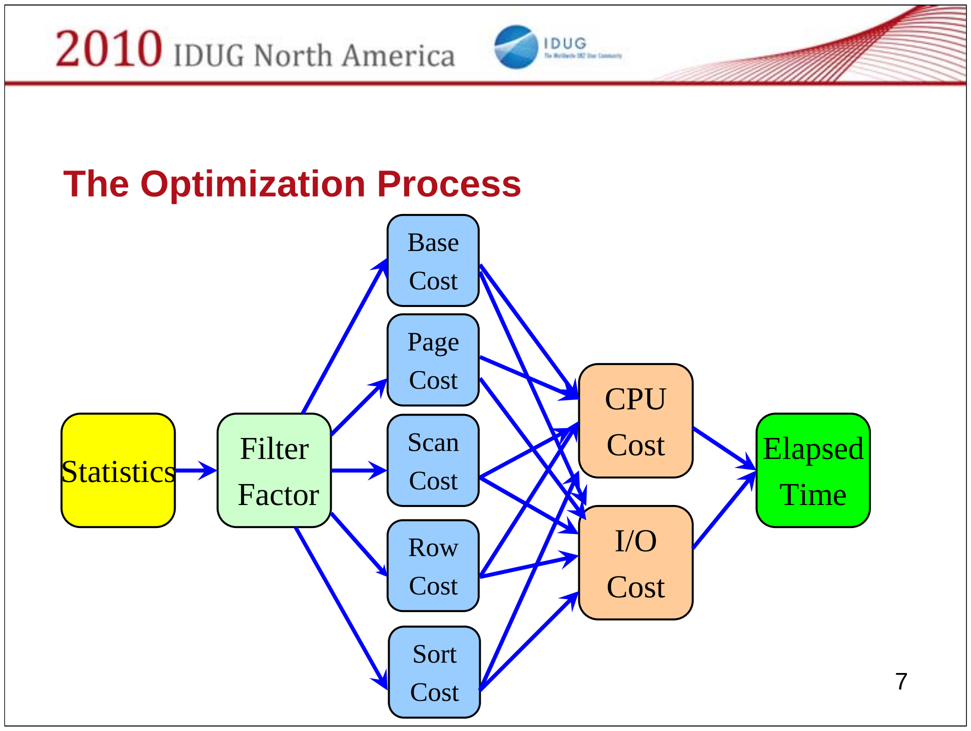
The basics of the program preparation process. We assume you are using the precompiler. When using the language co-processor, the flow is slightly different, but the resulting match in CONTOKEN at run time is identical.



2010 IDUG North America

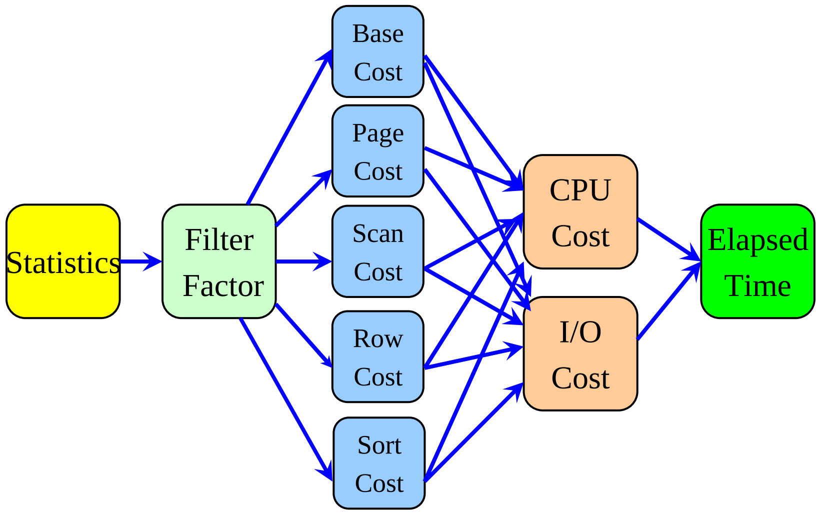
Optimizer basics

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



2010 IDUG North America

The Optimization Process

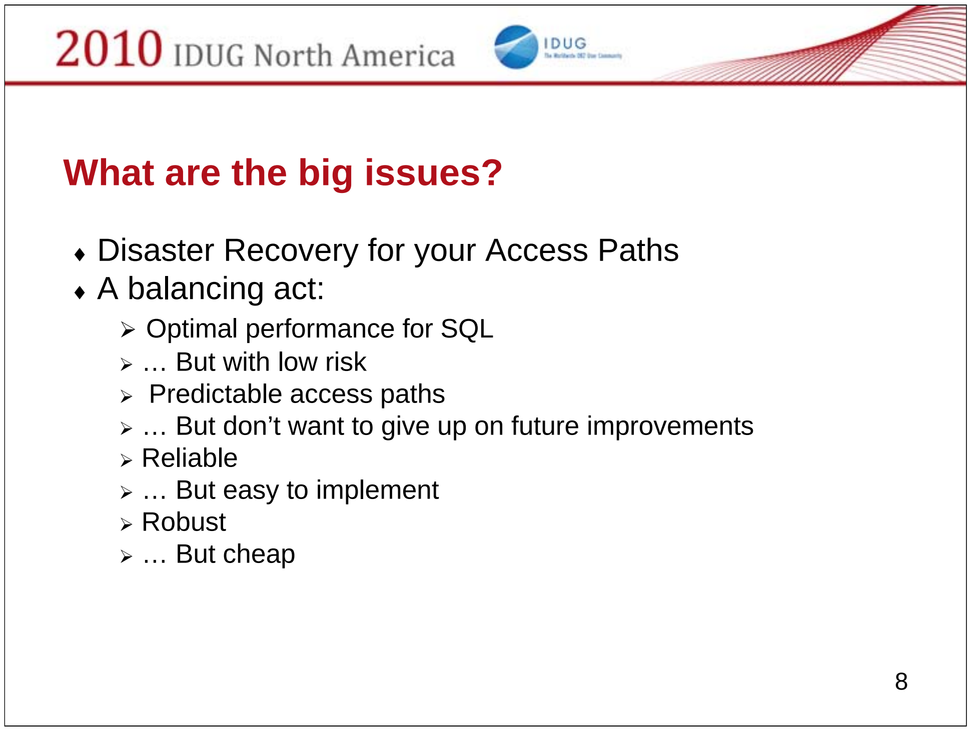


7

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)!

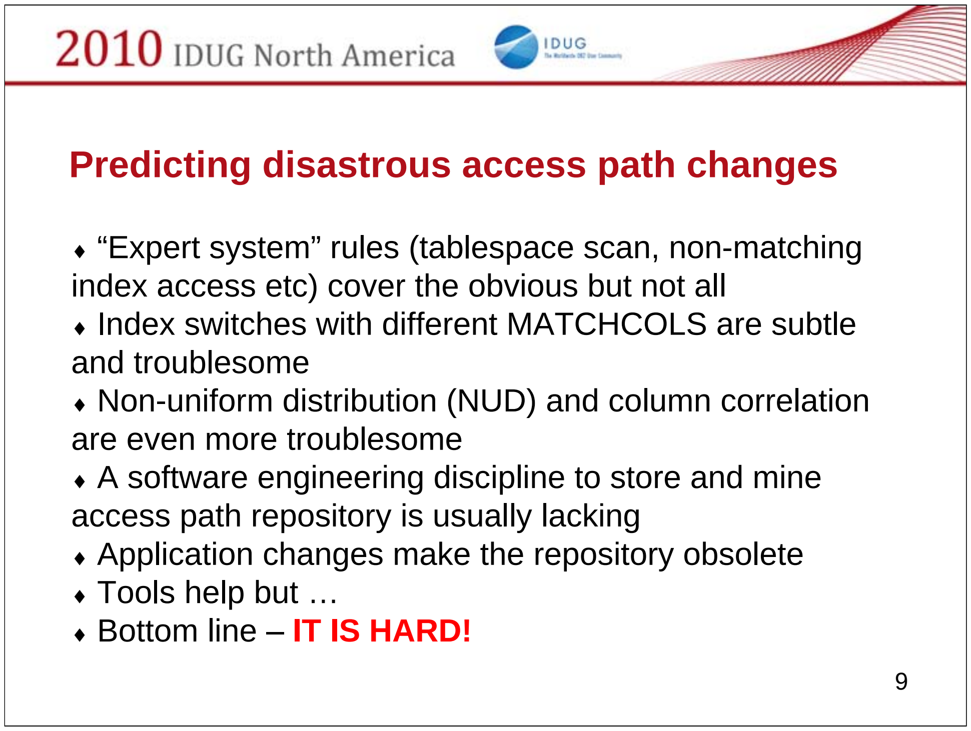

2010 IDUG North America

What are the big issues?

- Disaster Recovery for your Access Paths
- A balancing act:
  - Optimal performance for SQL
  - … But with low risk
  - Predictable access paths
  - … But don't want to give up on future improvements
  - Reliable
  - … But easy to implement
  - Robust
  - … But cheap

8

The conflicting goals we are trying to meet.

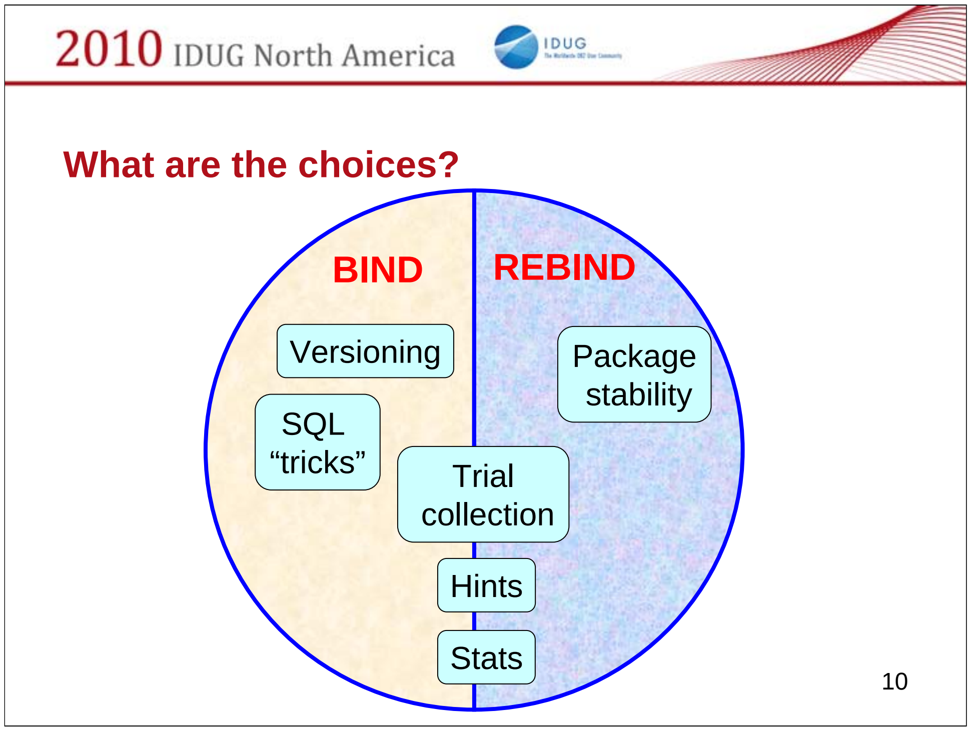

2010 IDUG North America

Predicting disastrous access path changes

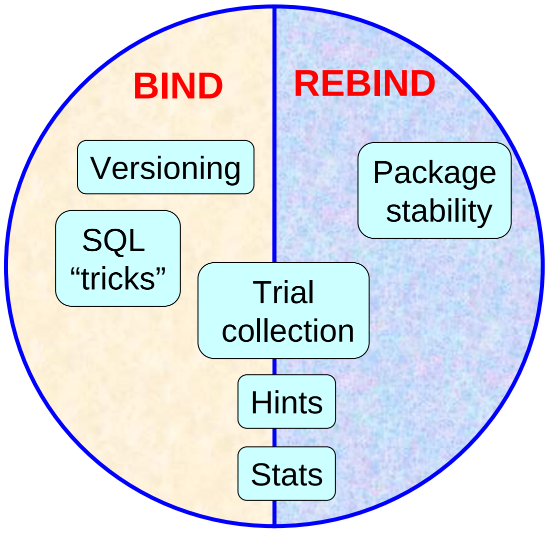
- ♦ "Expert system" rules (tablespace scan, non-matching index access etc) cover the obvious but not all
- ♦ Index switches with different MATCHCOLS are subtle and troublesome
- ♦ Non-uniform distribution (NUD) and column correlation are even more troublesome
- ♦ A software engineering discipline to store and mine access path repository is usually lacking
- ♦ Application changes make the repository obsolete
- ♦ Tools help but …
- ♦ Bottom line – **IT IS HARD!**

9

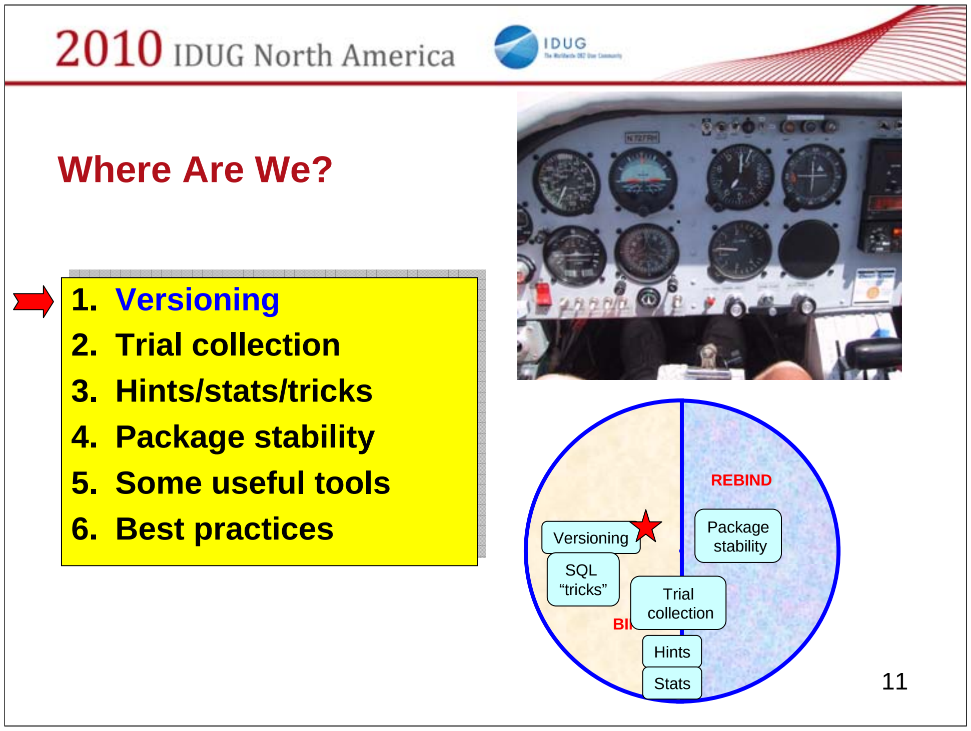
Bottom line – there is no easy way. You have to invest the resources to make it happen. See section 5 for information on some tools which may help with access path management.



What are the choices?



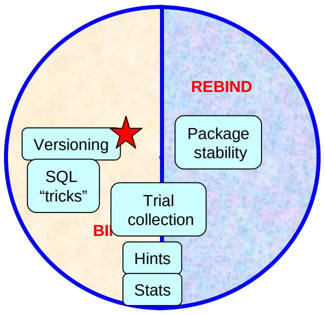
The various choices we will discuss in this presentation. Some apply to BINDs only, some to REBINDs only and some to both.



2010 IDUG North America

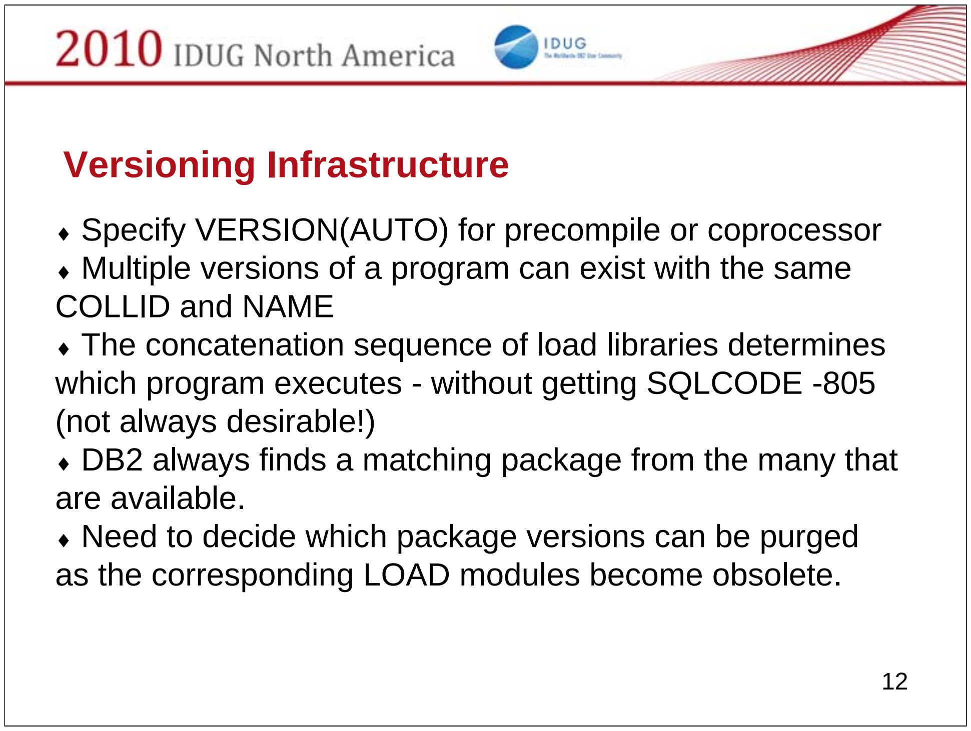
Where Are We?

1. Versioning
2. Trial collection
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. Best practices



11

The first option of using versions.

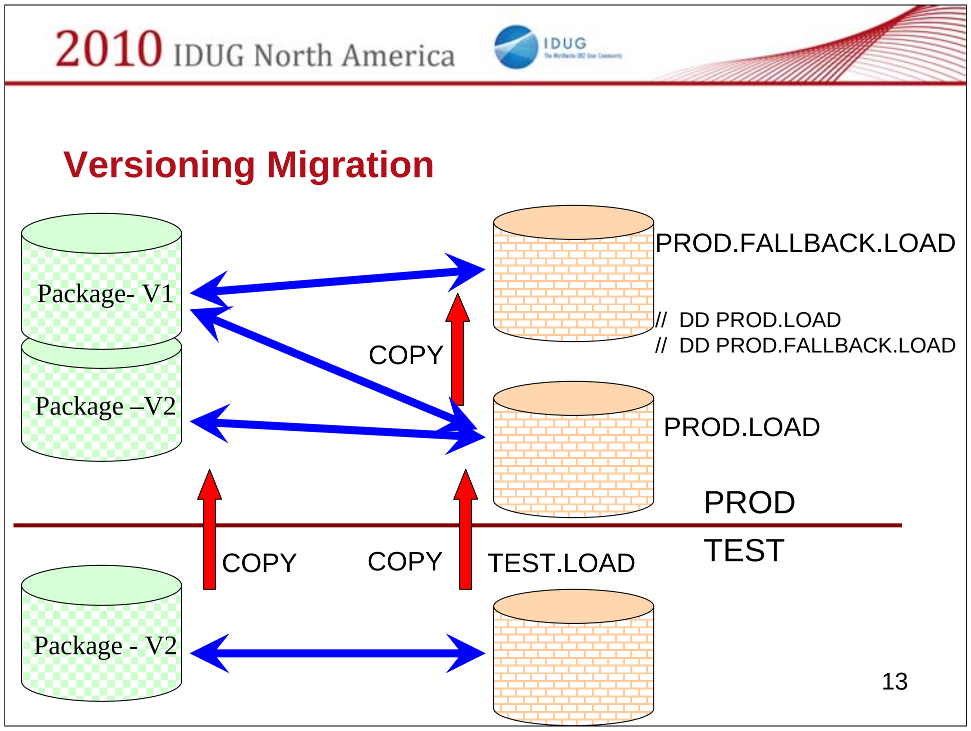

2010 IDUG North America

Versioning Infrastructure

- ♦ Specify VERSION(AUTO) for precompile or coprocessor
- ♦ Multiple versions of a program can exist with the same COLLID and NAME
- ♦ The concatenation sequence of load libraries determines which program executes - without getting SQLCODE -805 (not always desirable!)
- ♦ DB2 always finds a matching package from the many that are available.
- ♦ Need to decide which package versions can be purged as the corresponding LOAD modules become obsolete.

12

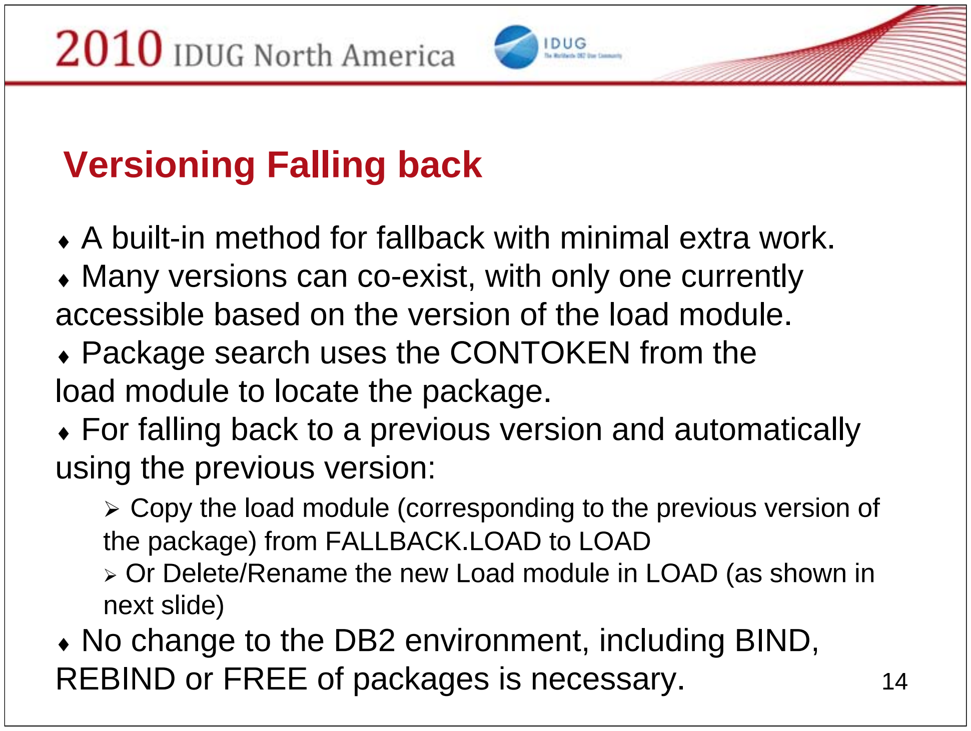
What is needed to implement versions.



2010 IDUG North America

Versioning Migration

How the program migration works when using versions.

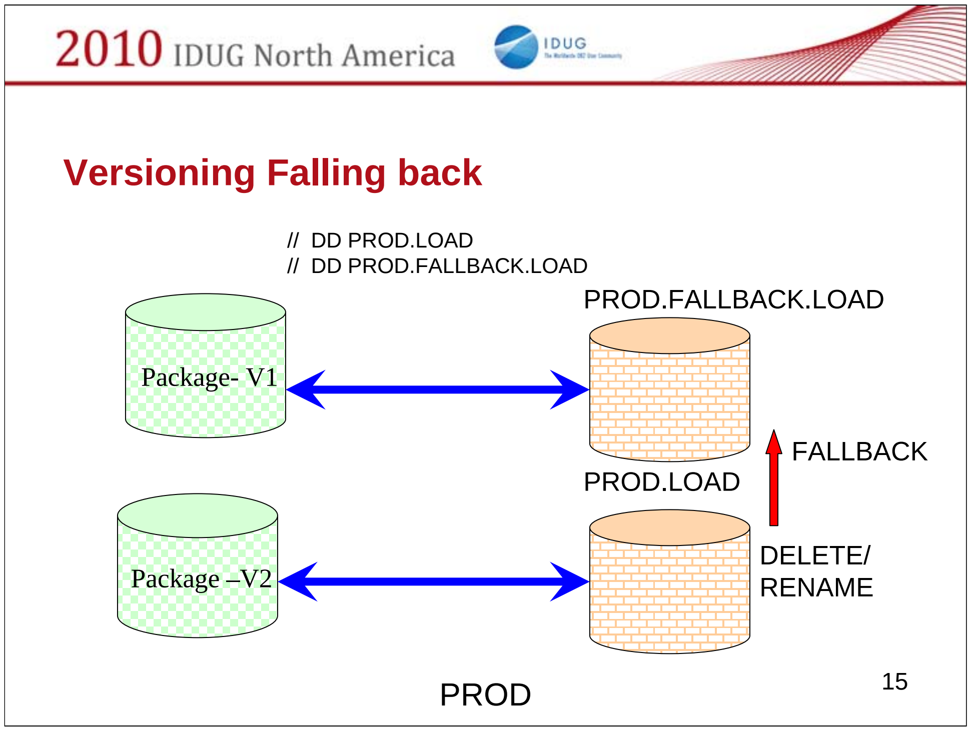

2010 IDUG North America

Versioning Falling back

- ♦ A built-in method for fallback with minimal extra work.
- ♦ Many versions can co-exist, with only one currently accessible based on the version of the load module.
- ♦ Package search uses the CONTOKEN from the load module to locate the package.
- ♦ For falling back to a previous version and automatically using the previous version:
  - ➢ Copy the load module (corresponding to the previous version of the package) from FALLBACK.LOAD to LOAD
  - ➢ Or Delete/Rename the new Load module in LOAD (as shown in next slide)
- ♦ No change to the DB2 environment, including BIND, REBIND or FREE of packages is necessary.

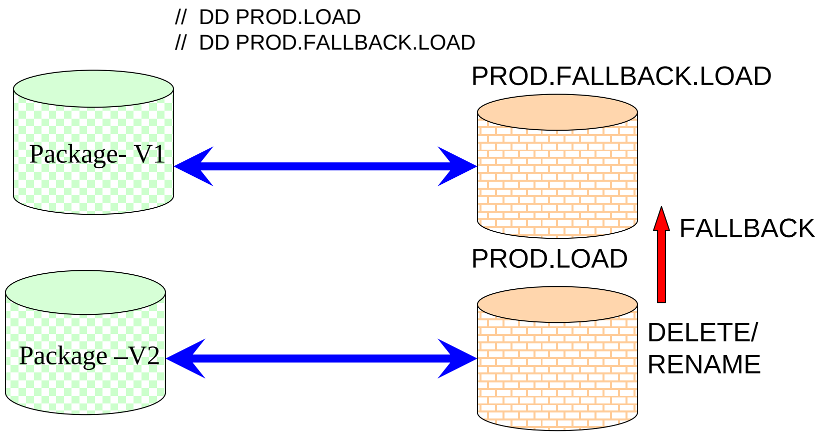
14

… and how to fallback.



2010 IDUG North America

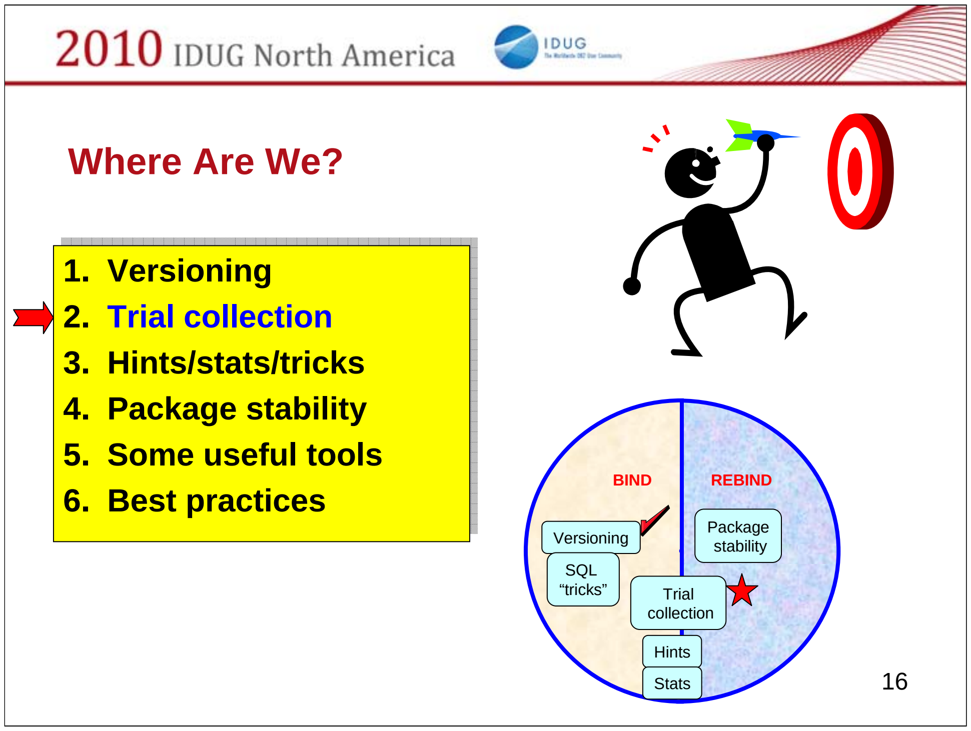
Versioning Falling back



PROD

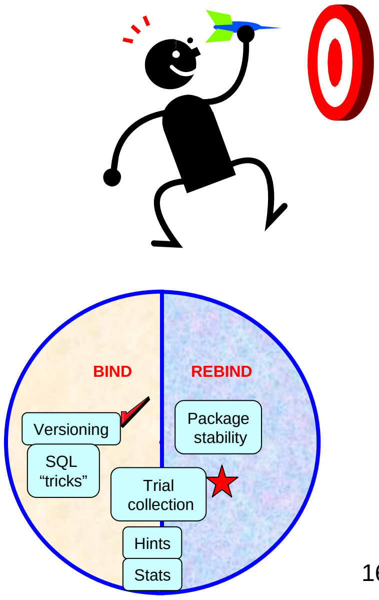
15

Fallback scenario.

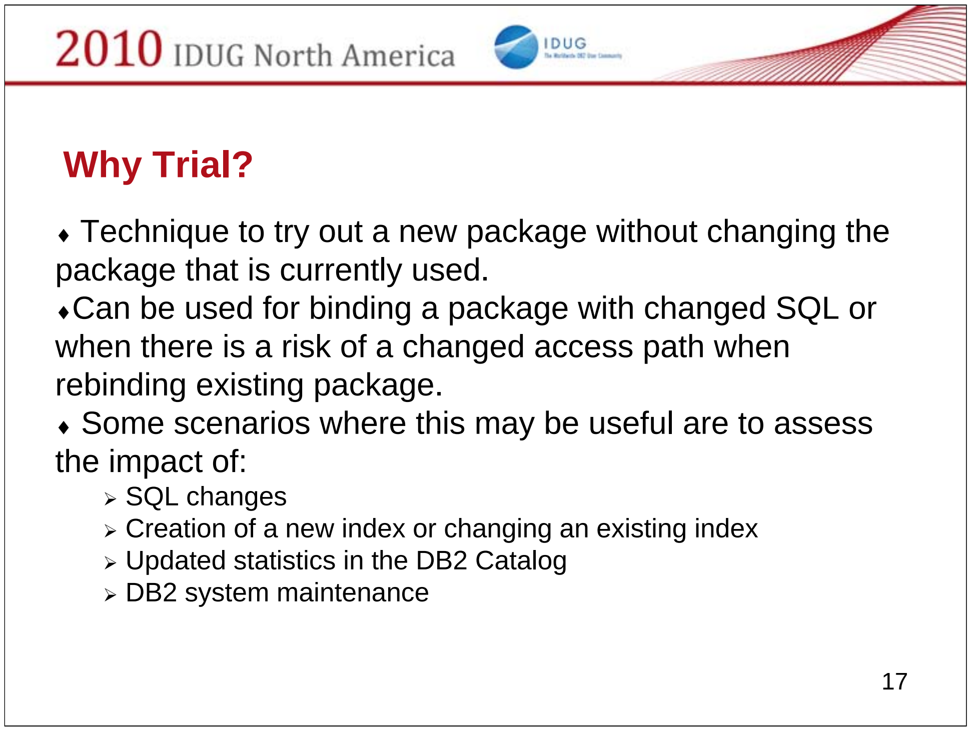


Where Are We?

1. Versioning
2. **Trial collection**
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. Best practices



The second option – using a trial collection.



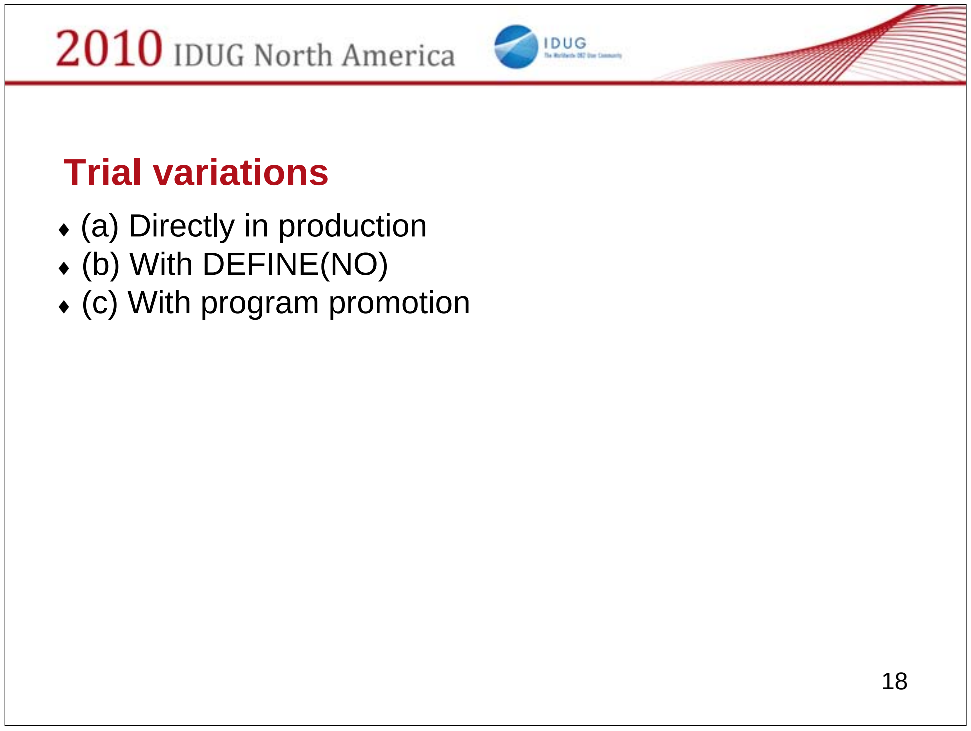
2010 IDUG North America

Why Trial?

- ♦ Technique to try out a new package without changing the package that is currently used.
- ♦Can be used for binding a package with changed SQL or when there is a risk of a changed access path when rebinding existing package.
- ♦ Some scenarios where this may be useful are to assess the impact of:
  - ➢ SQL changes
  - ➢ Creation of a new index or changing an existing index
  - ➢ Updated statistics in the DB2 Catalog
  - ➢ DB2 system maintenance

17

The business driver for considering a trial collection.



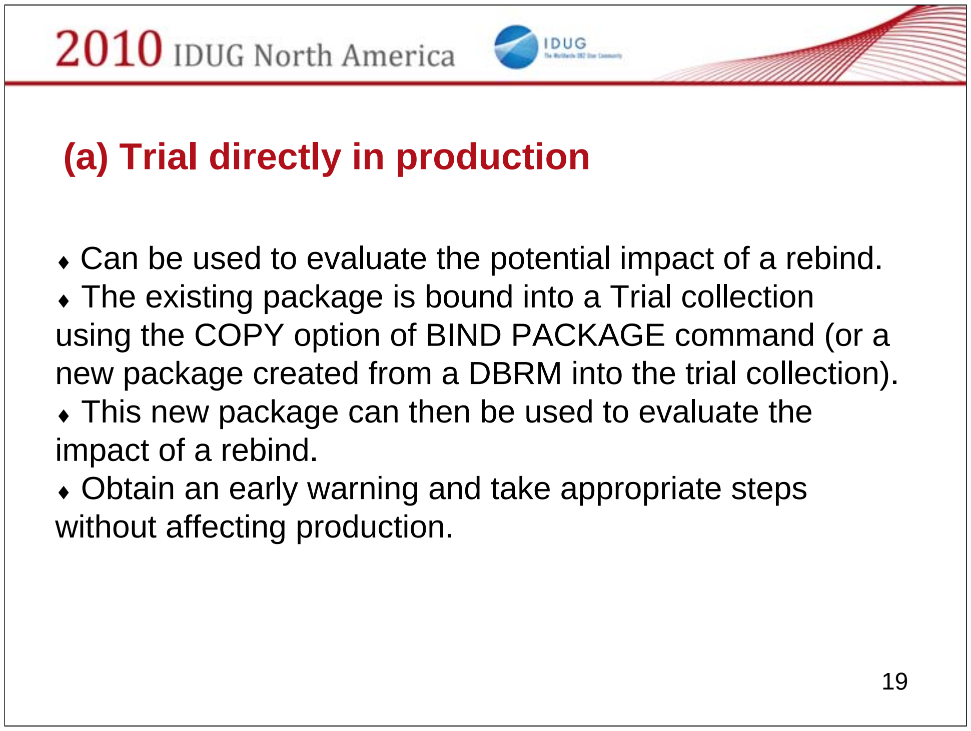
2010 IDUG North America

Trial variations

- (a) Directly in production
- (b) With DEFINE(NO)
- (c) With program promotion

18

The variations of using a trial collection.



(a) Trial directly in production

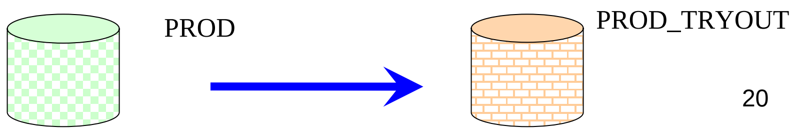
- Can be used to evaluate the potential impact of a rebind.
- The existing package is bound into a Trial collection using the COPY option of BIND PACKAGE command (or a new package created from a DBRM into the trial collection).
- This new package can then be used to evaluate the impact of a rebind.
- Obtain an early warning and take appropriate steps without affecting production.

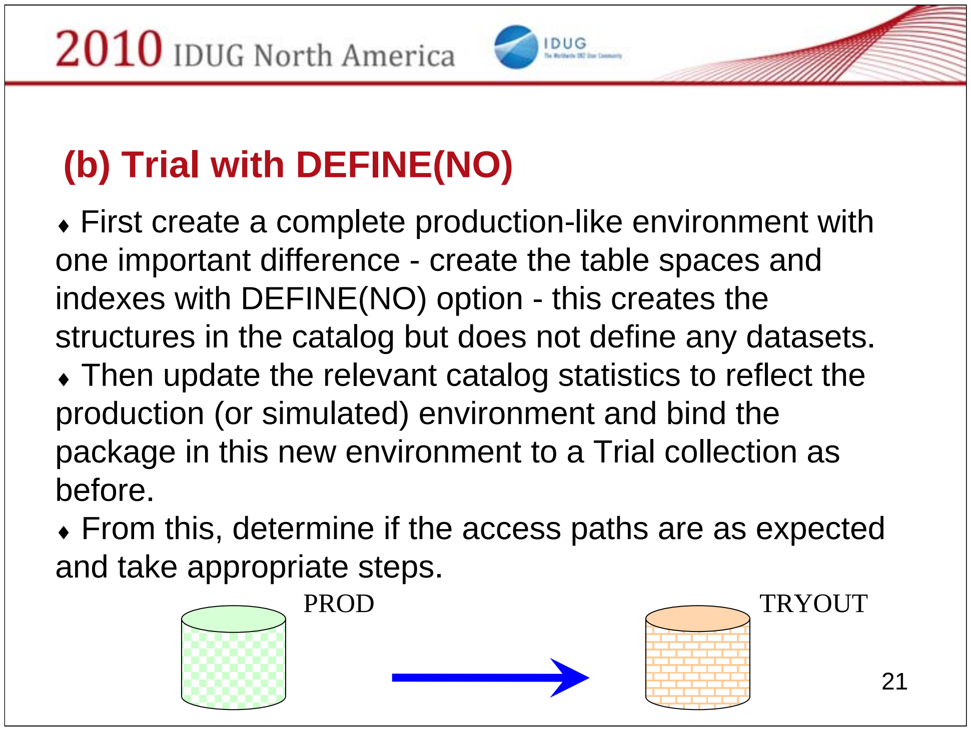
The first variation.

(a) Trial directly in production

Example:

- To test the impact of a newly created index. The PROD package is bound into the PROD_TRYOUT collection.
- Re-optimization occurs during bind time and may include a new access path that uses the new index.
- Determine whether the newly created index is used, then review the access path to assess whether it is likely to provide acceptable performance in the production environment.

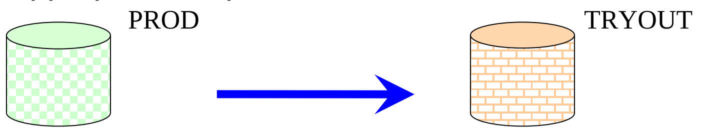




2010 IDUG North America

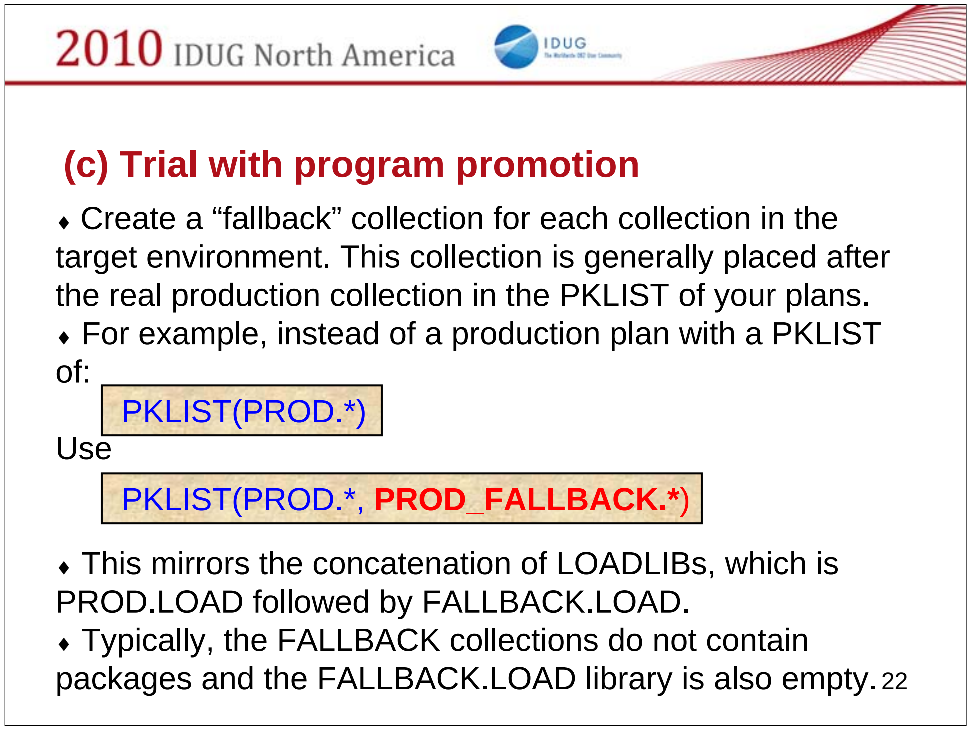
(b) Trial with DEFINE(NO)

- First create a complete production-like environment with one important difference - create the table spaces and indexes with DEFINE(NO) option - this creates the structures in the catalog but does not define any datasets.
- Then update the relevant catalog statistics to reflect the production (or simulated) environment and bind the package in this new environment to a Trial collection as before.
- From this, determine if the access paths are as expected and take appropriate steps.



21

The second variation. The catalog stats must be updated correctly and consistently.



2010 IDUG North America

(c) Trial with program promotion

- Create a "fallback" collection for each collection in the target environment. This collection is generally placed after the real production collection in the PKLIST of your plans.
- For example, instead of a production plan with a PKLIST of:

```
PKLIST(PROD.*)
```

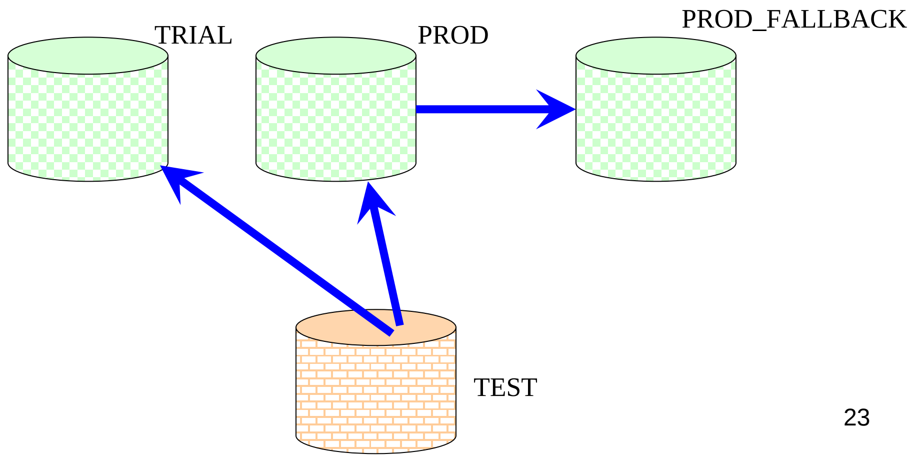
Use

```
PKLIST(PROD.*, PROD_FALLBACK.*)
```

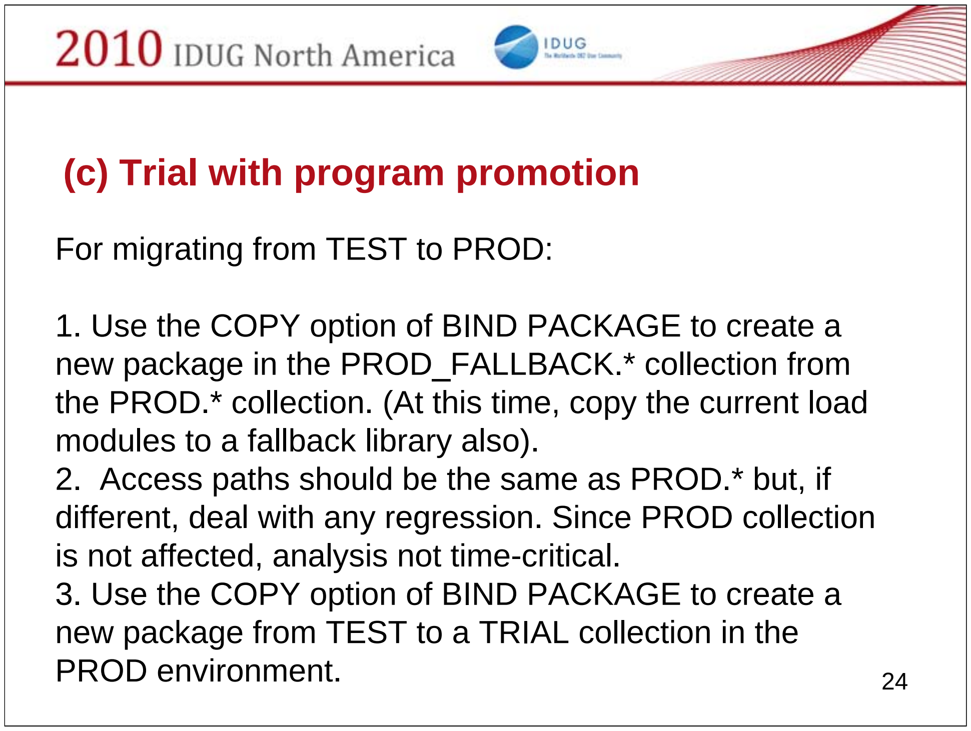
- This mirrors the concatenation of LOADLIBs, which is PROD.LOAD followed by FALLBACK.LOAD.
- Typically, the FALLBACK collections do not contain packages and the FALLBACK.LOAD library is also empty. 22

..and the third. This integrates the approach in your program migration scheme.

(c) Trial with program promotion - Overview



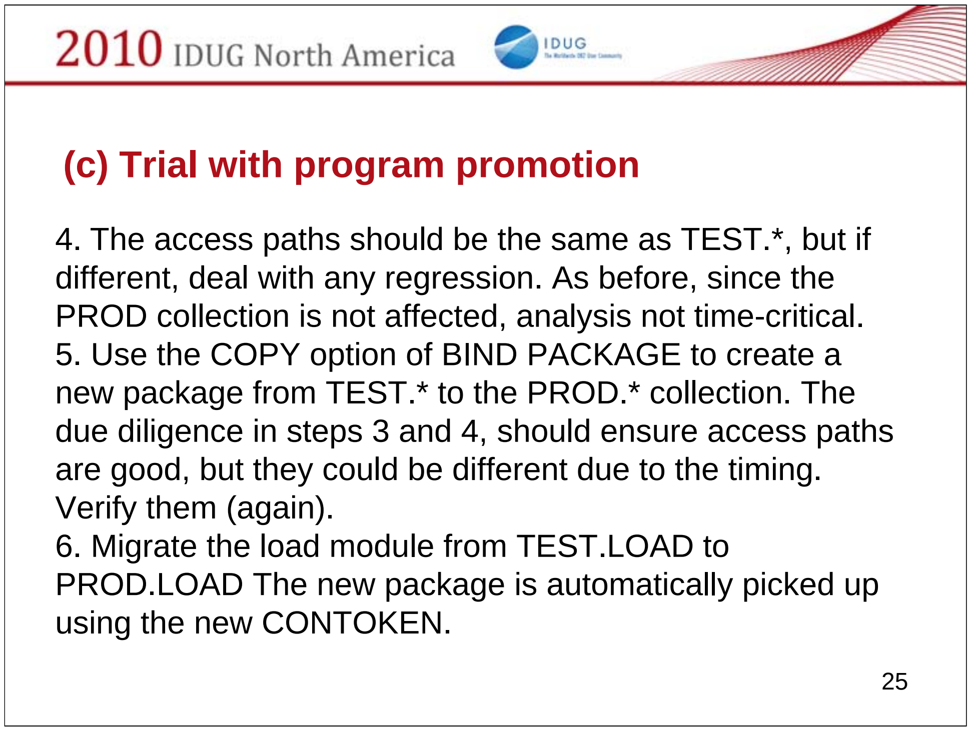
23



(c) Trial with program promotion

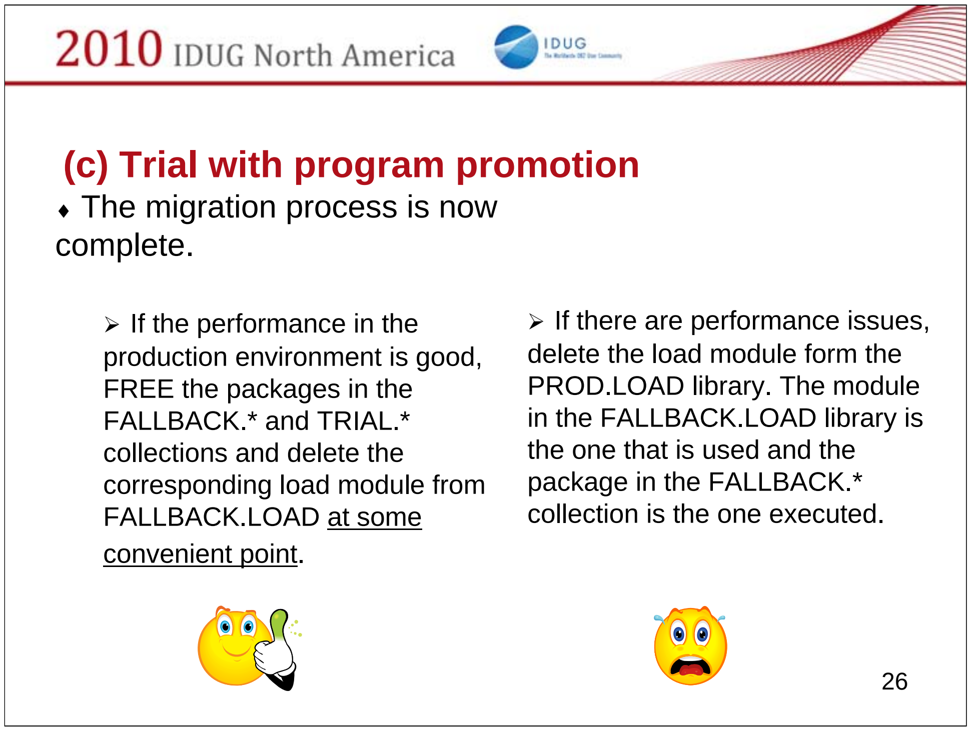
For migrating from TEST to PROD:

1. Use the COPY option of BIND PACKAGE to create a new package in the PROD_FALLBACK.* collection from the PROD.* collection. (At this time, copy the current load modules to a fallback library also).
2. Access paths should be the same as PROD.* but, if different, deal with any regression. Since PROD collection is not affected, analysis not time-critical.
3. Use the COPY option of BIND PACKAGE to create a new package from TEST to a TRIAL collection in the PROD environment.



(c) Trial with program promotion

4. The access paths should be the same as TEST.*, but if different, deal with any regression. As before, since the PROD collection is not affected, analysis not time-critical.
5. Use the COPY option of BIND PACKAGE to create a new package from TEST.* to the PROD.* collection. The due diligence in steps 3 and 4, should ensure access paths are good, but they could be different due to the timing. Verify them (again).
6. Migrate the load module from TEST.LOAD to PROD.LOAD The new package is automatically picked up using the new CONTOKEN.



2010 IDUG North America

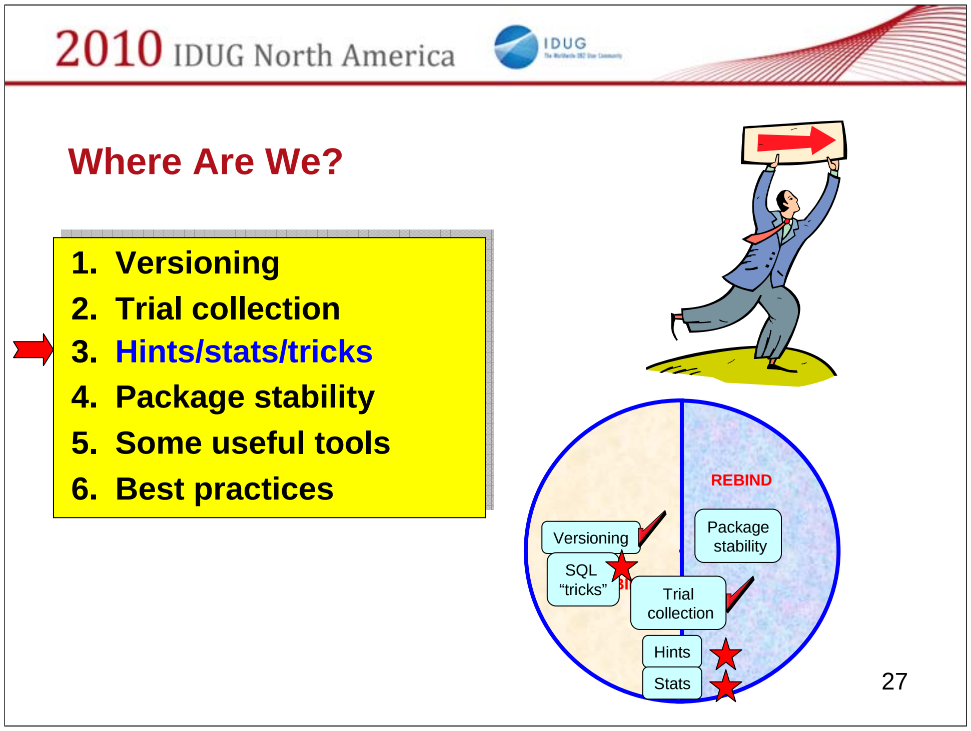
(c) Trial with program promotion

- The migration process is now complete.

  - If the performance in the production environment is good, FREE the packages in the FALLBACK.* and TRIAL.* collections and delete the corresponding load module from FALLBACK.LOAD at some convenient point.
  - If there are performance issues, delete the load module form the PROD.LOAD library. The module in the FALLBACK.LOAD library is the one that is used and the package in the FALLBACK.* collection is the one executed.

26

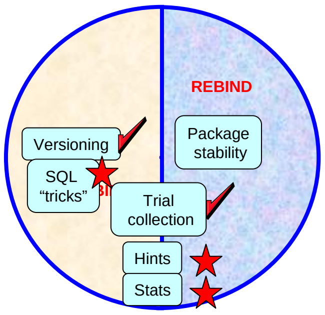
The critical decision point and how to fallback if not successful.



2010 IDUG North America

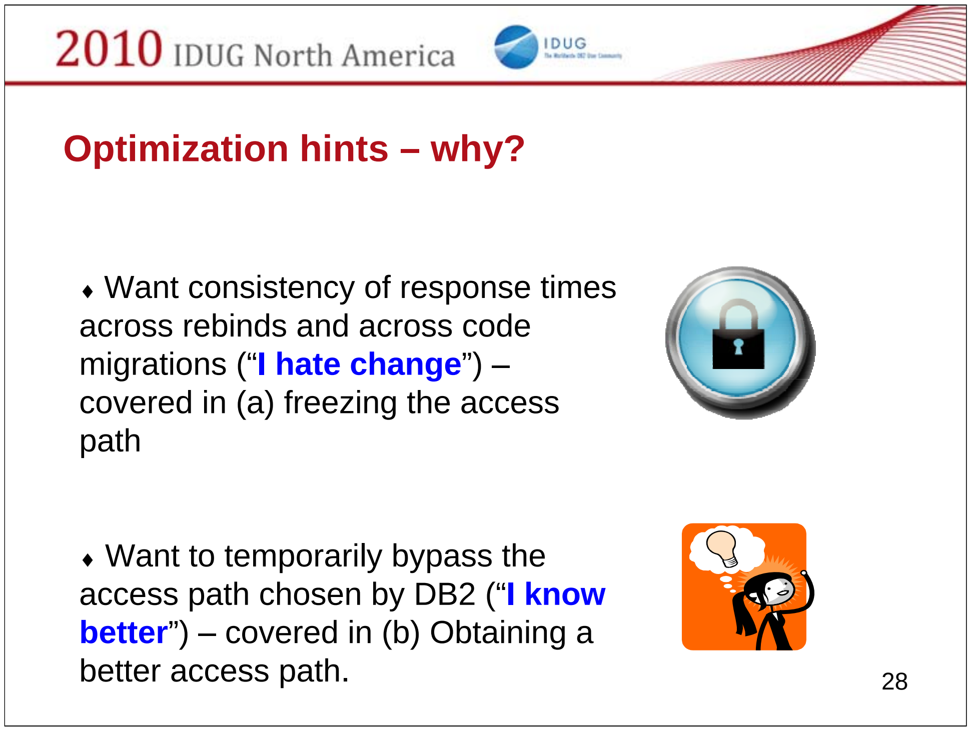
Where Are We?

1. Versioning
2. Trial collection
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. Best practices



27

Why you may need to influencing the access path and how to do it.



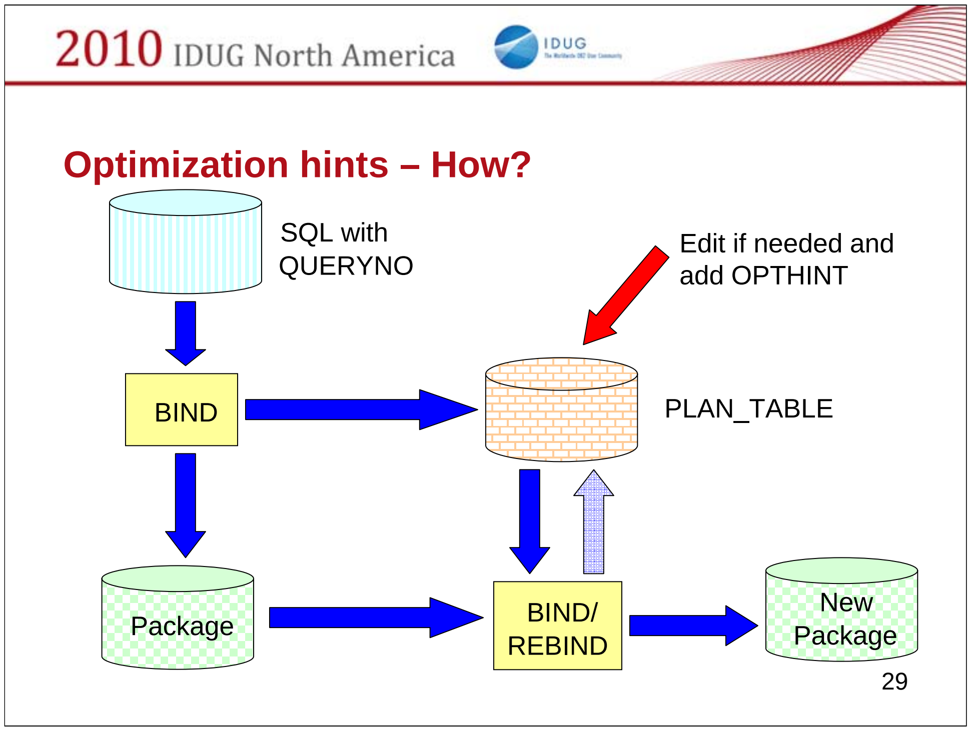
2010 IDUG North America

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.

28

The business drivers for hints.

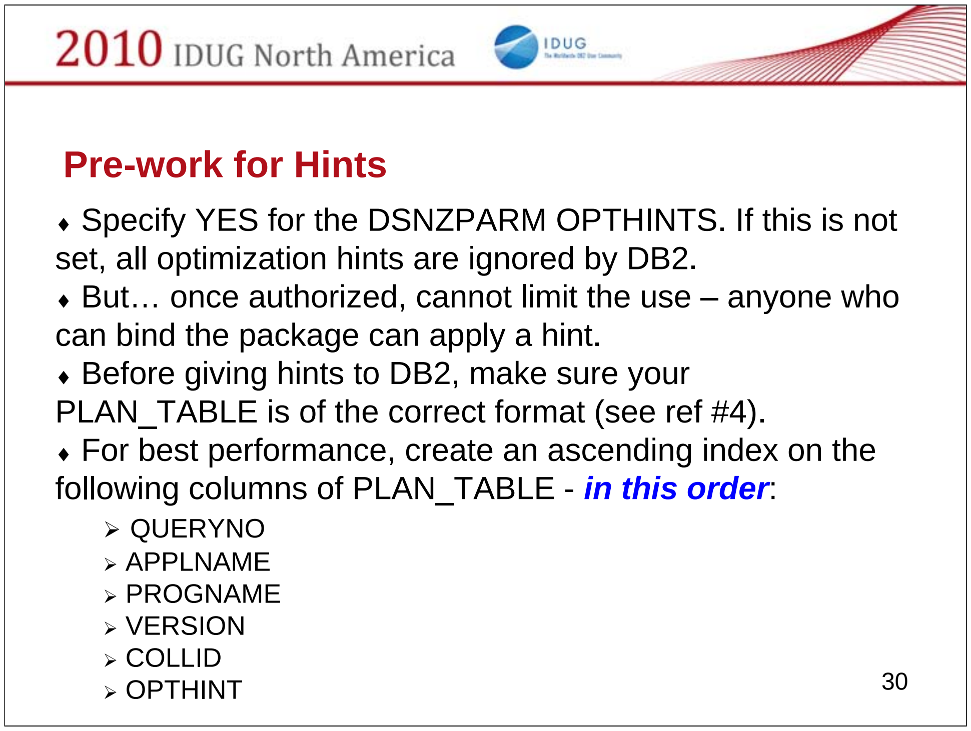


2010 IDUG North America

Optimization hints – How?

29

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.

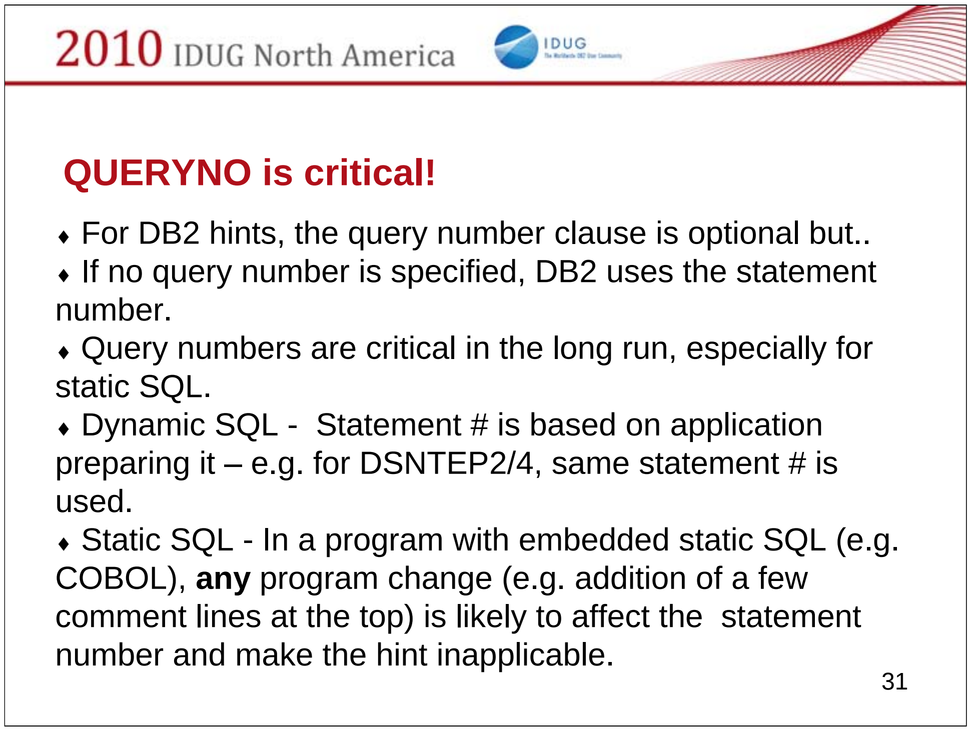

2010 IDUG North America

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

30

What is needed to implement hints.

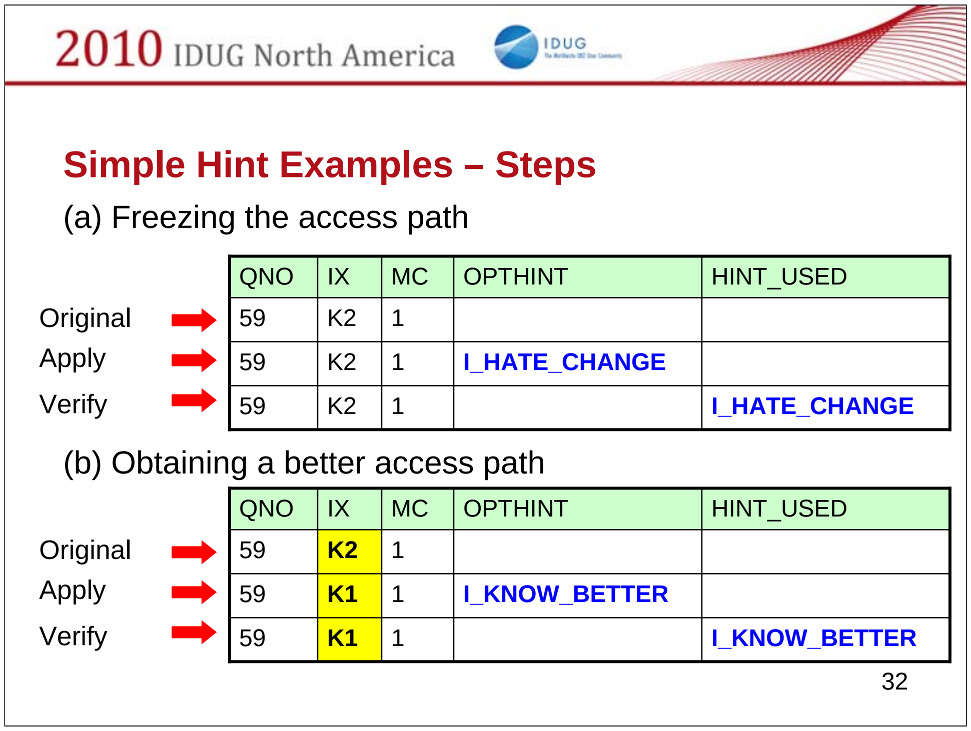

2010 IDUG North America

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.

31

QUERYNO is critical!!



2010 IDUG North America

Simple Hint Examples – Steps

(a) Freezing the access path

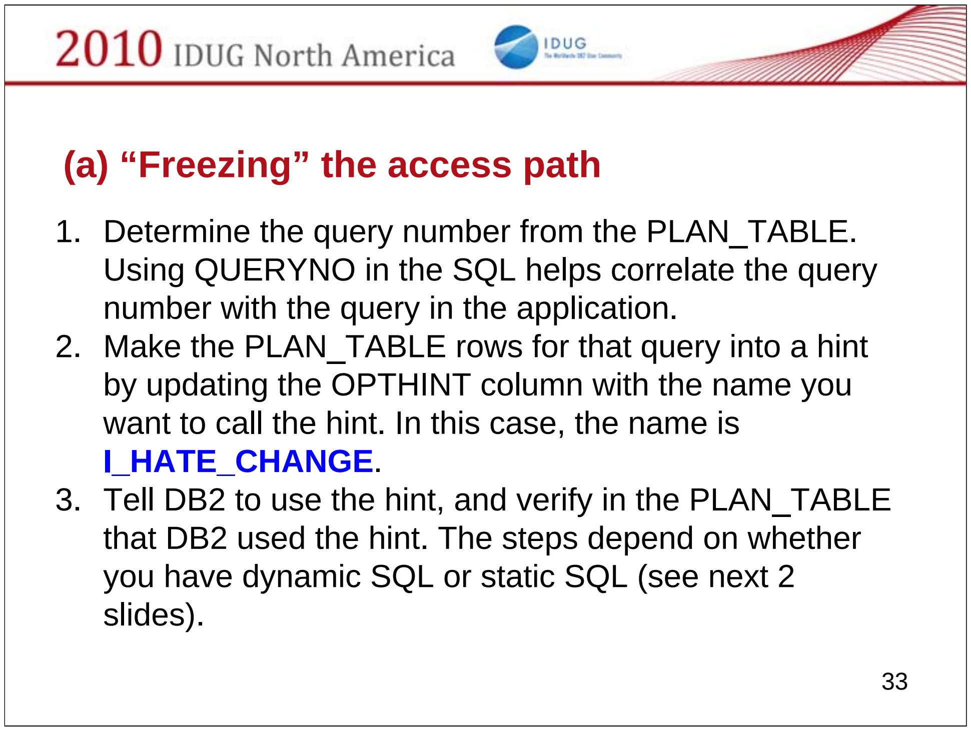
	QNO	IX	MC	OPTHINT	HINT_USED
Original	59	K2	1		
Apply	59	K2	1	**I_HATE_CHANGE**	
Verify	59	K2	1		**I_HATE_CHANGE**

(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**

32

What the plan_table looks like before and after.

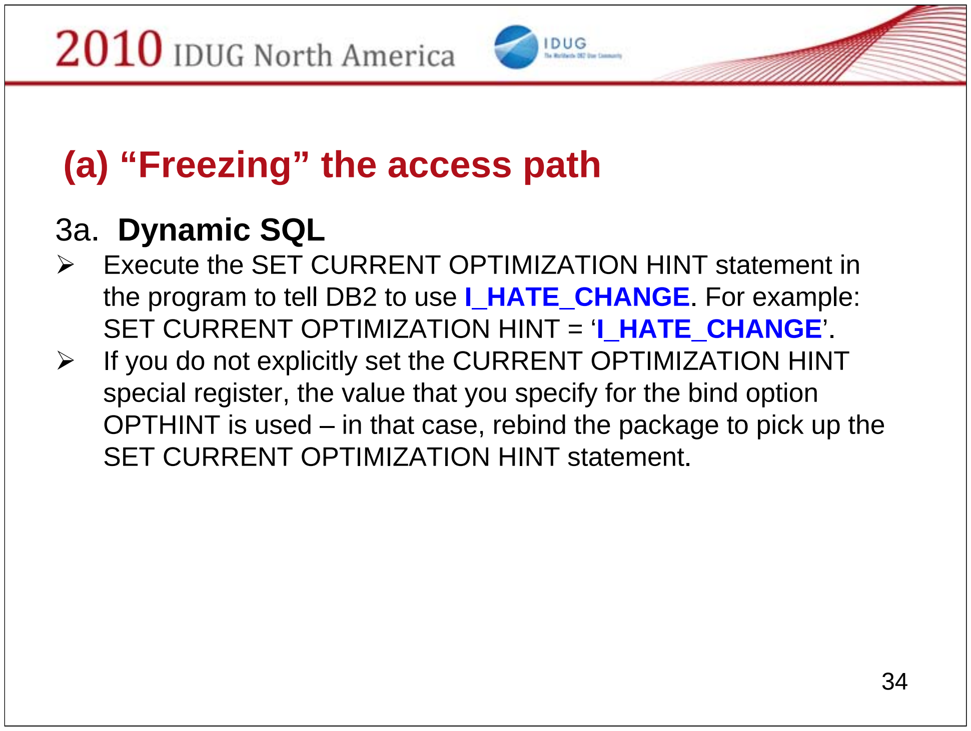

2010 IDUG North America

(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).

33

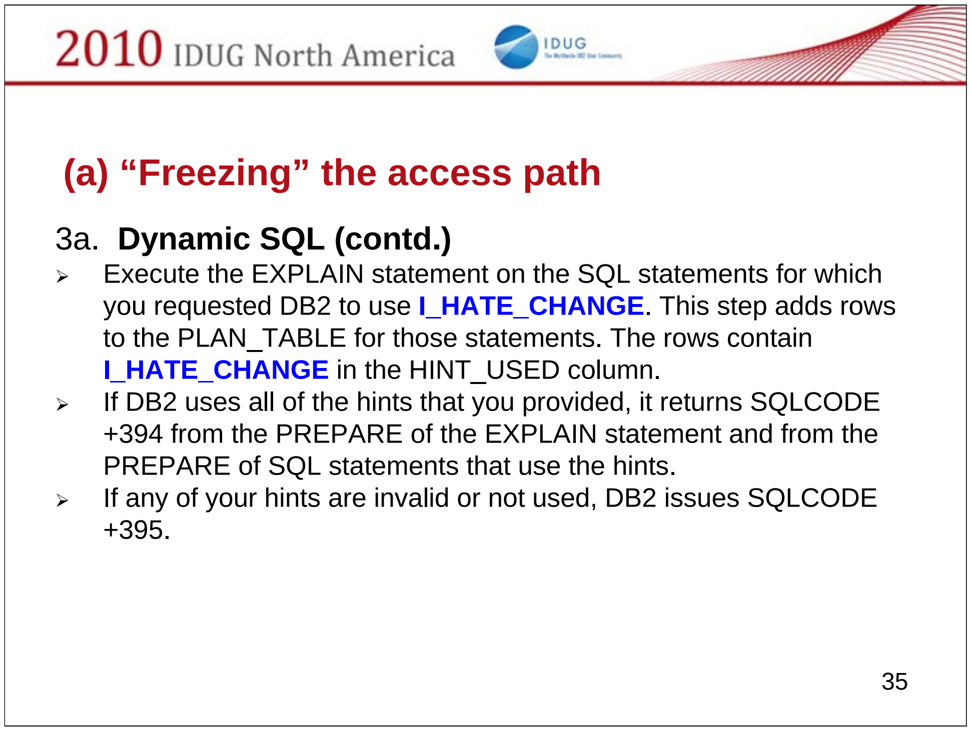
For “freezing” a good access path.



(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.


2010 IDUG North America

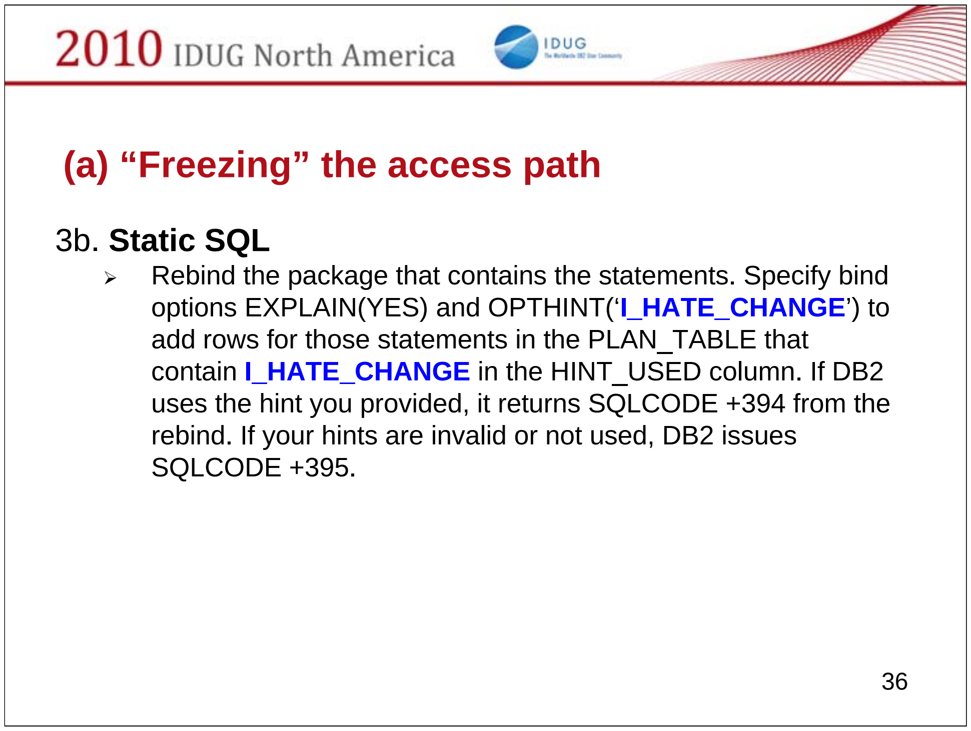
(a) "Freezing" the access path

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.

35

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.


2010 IDUG North America

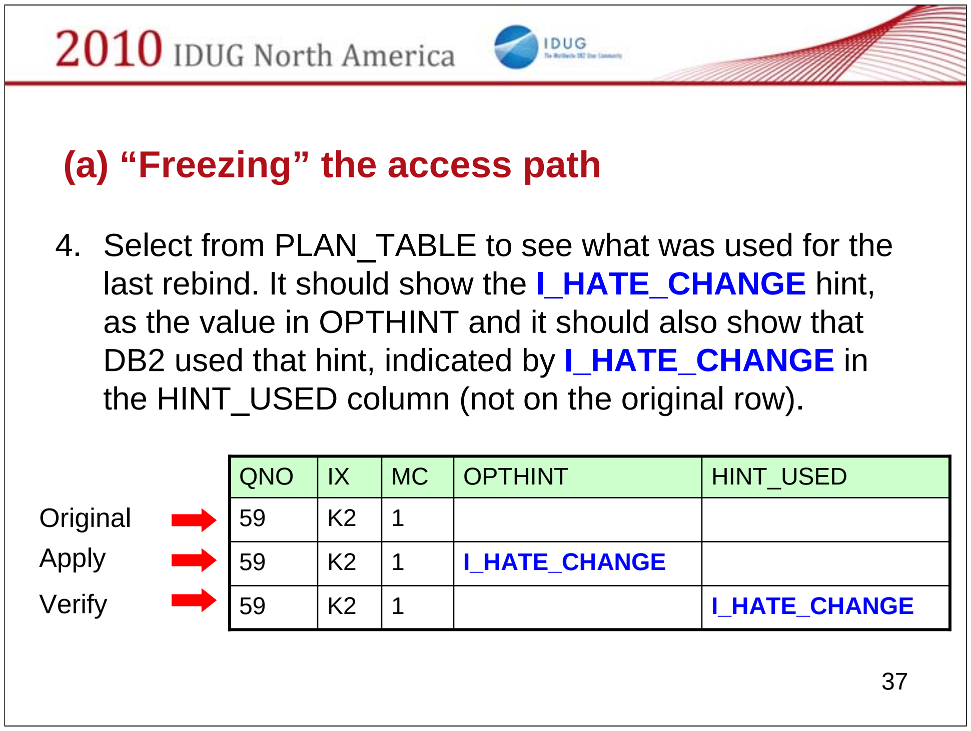
(a) “Freezing” the access path

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.

36

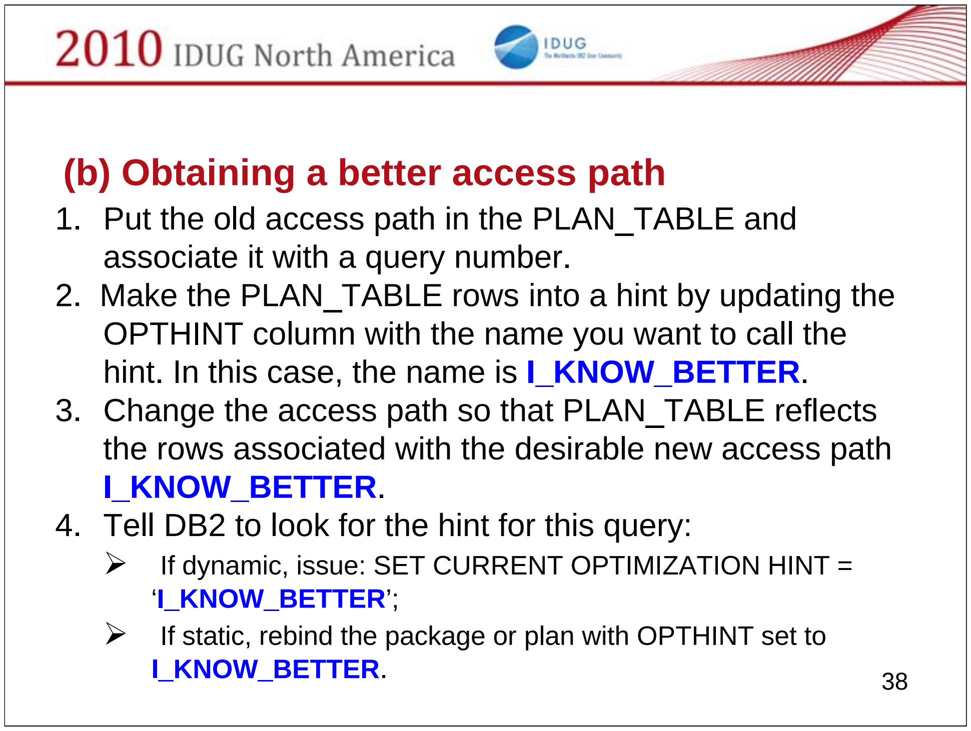
If DB2 issues SQLCODE +395 a reason code is also returned. About 40 reason codes are possible (e.g. **15** = Specified index cannot be used as requested, **19** = Nested loop join cannot be done as requested etc.). This is very useful in diagnosing what the actual error is.



(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**



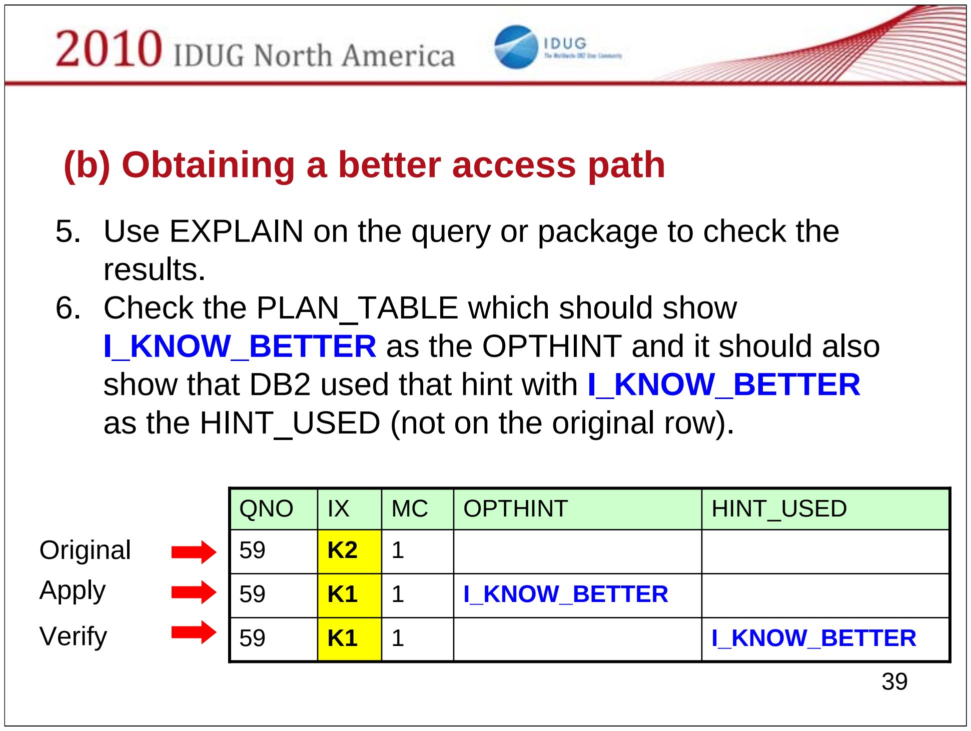
2010 IDUG North America

(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**.

38

… and for obtaining a better one.



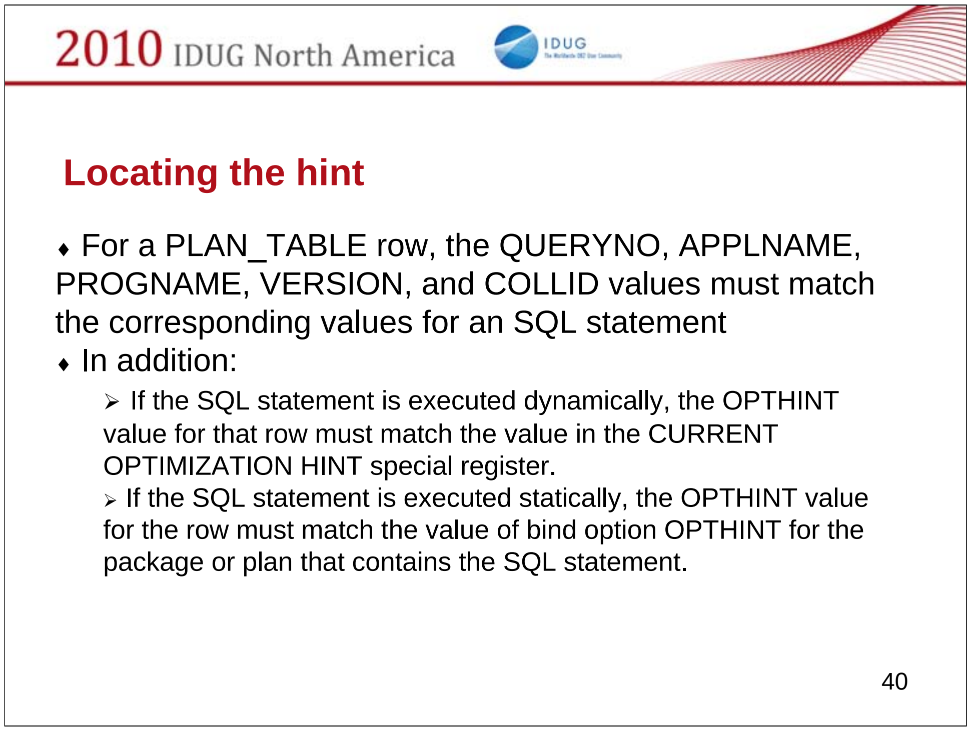
2010 IDUG North America

(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**

39



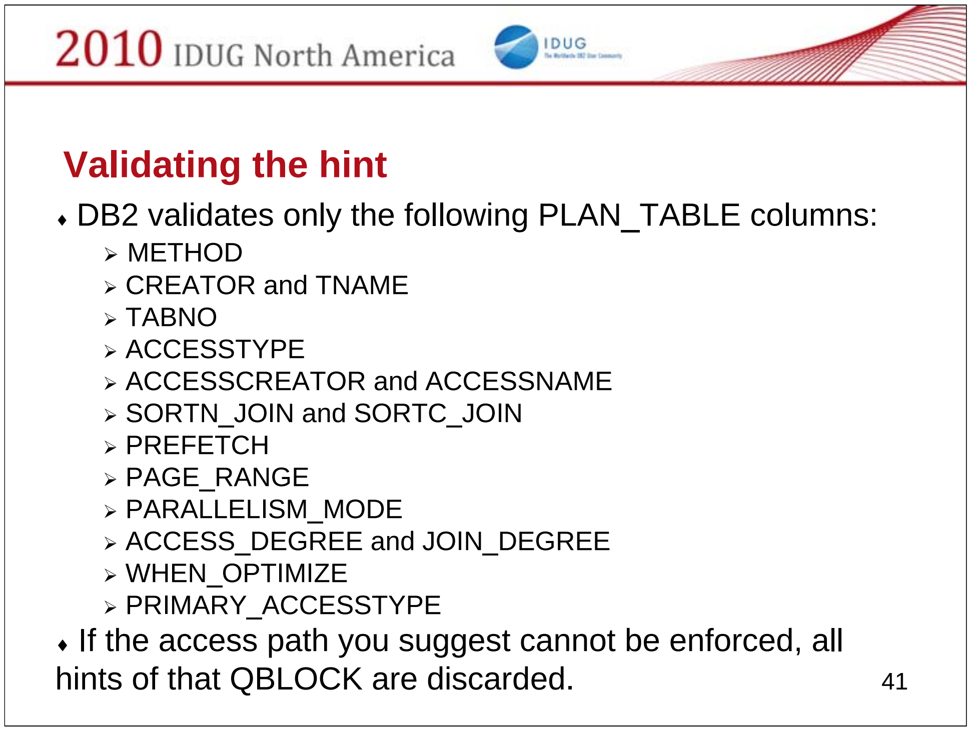
2010 IDUG North America

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.

40

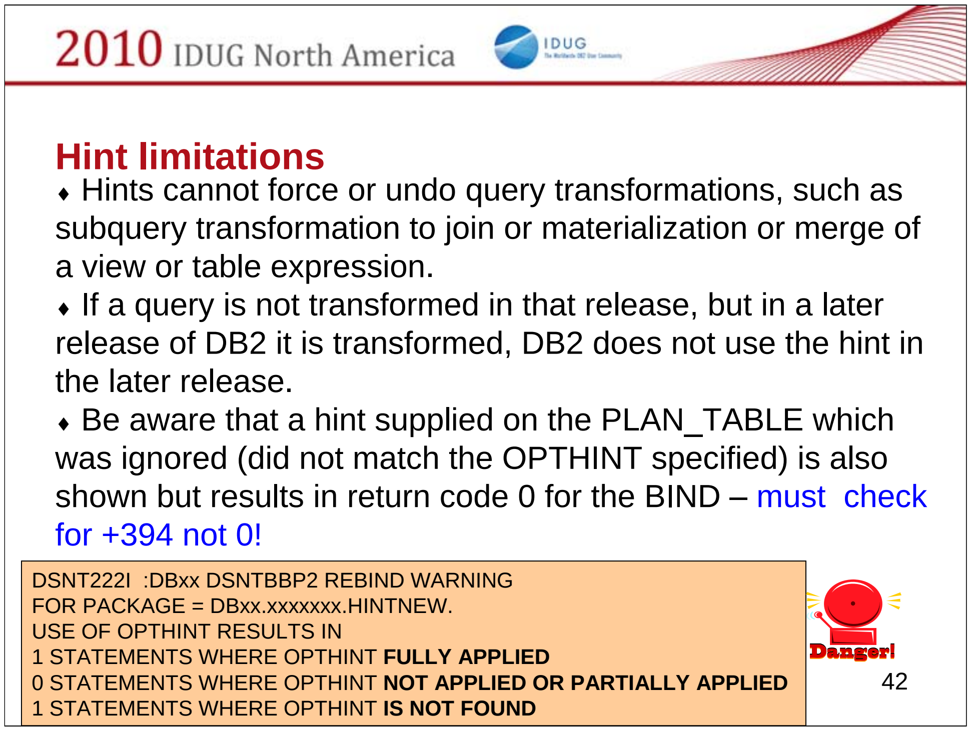
How does DB2 locate the hint in the plan_table?



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.

Rules for validating the hint.



2010 IDUG North America

Hint limitations

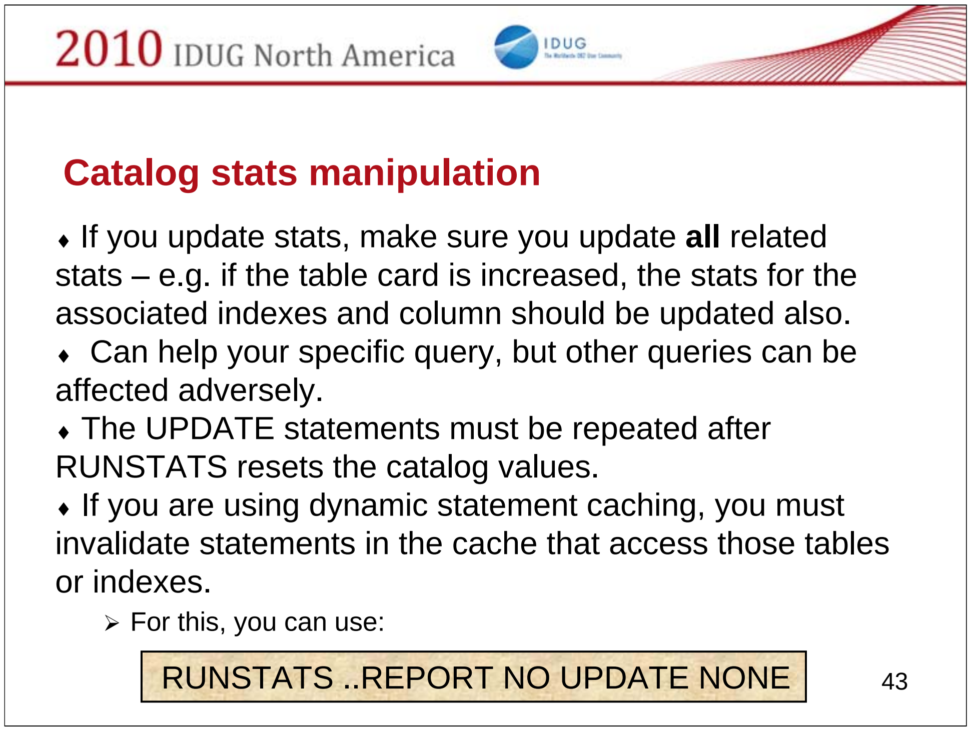
- Hints cannot force or undo query transformations, such as subquery transformation to join or materialization or merge of a view or table expression.
- If a query is not transformed in that release, but in a later release of DB2 it is transformed, DB2 does not use the hint in the later release.
- Be 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
```

Danger!

42

Some of the limitations.



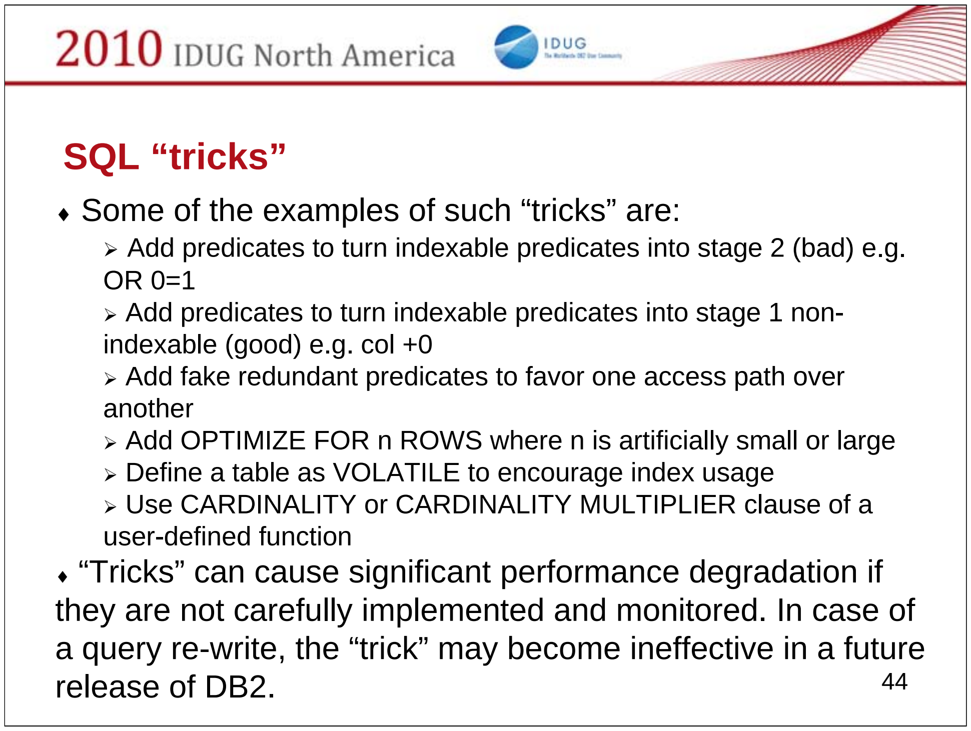
Catalog stats manipulation

- 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
```

43

Manipulating the catalog stats to influence the access path.

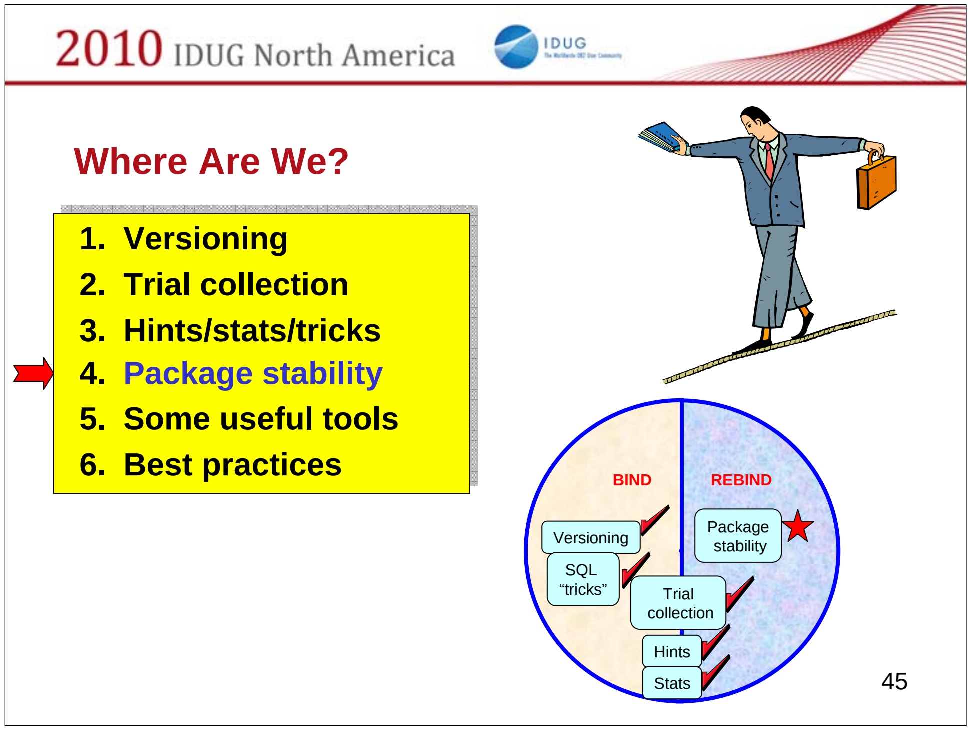

2010 IDUG North America

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.

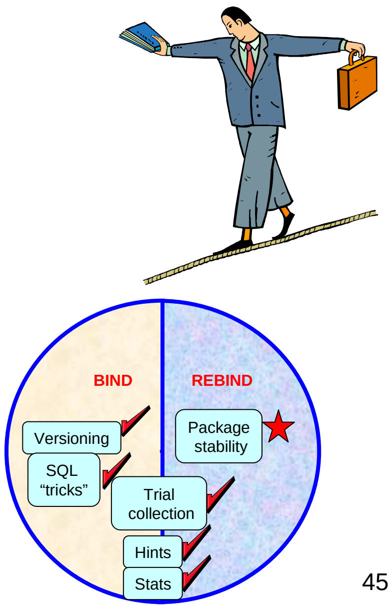
44

Using “tricks” to influence the access path.

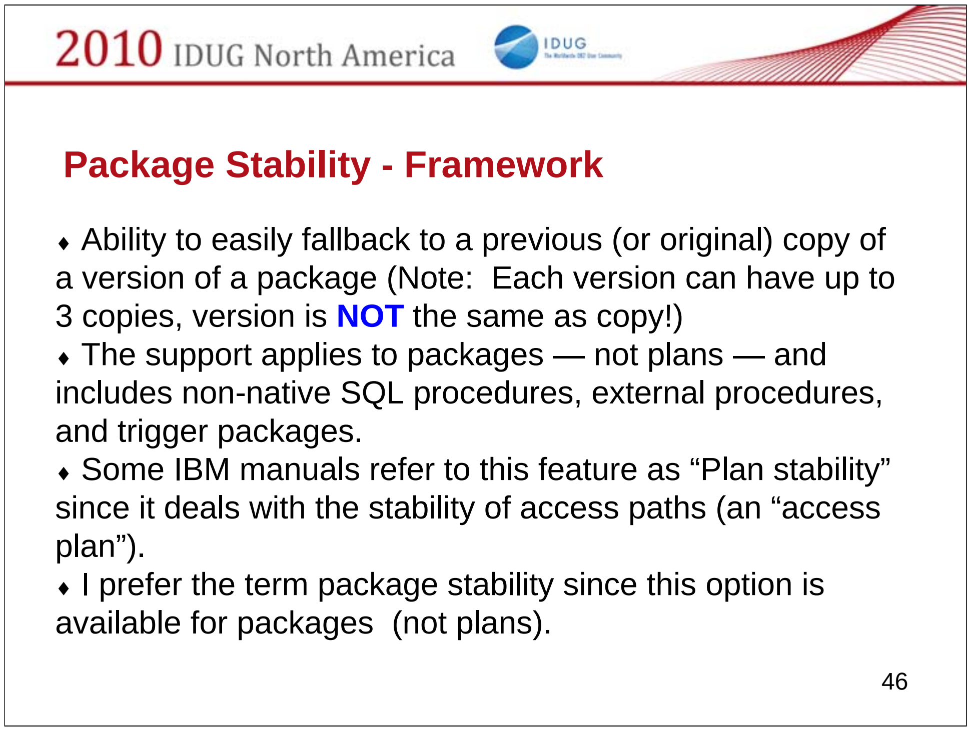


Where Are We?

1. Versioning
2. Trial collection
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. Best practices



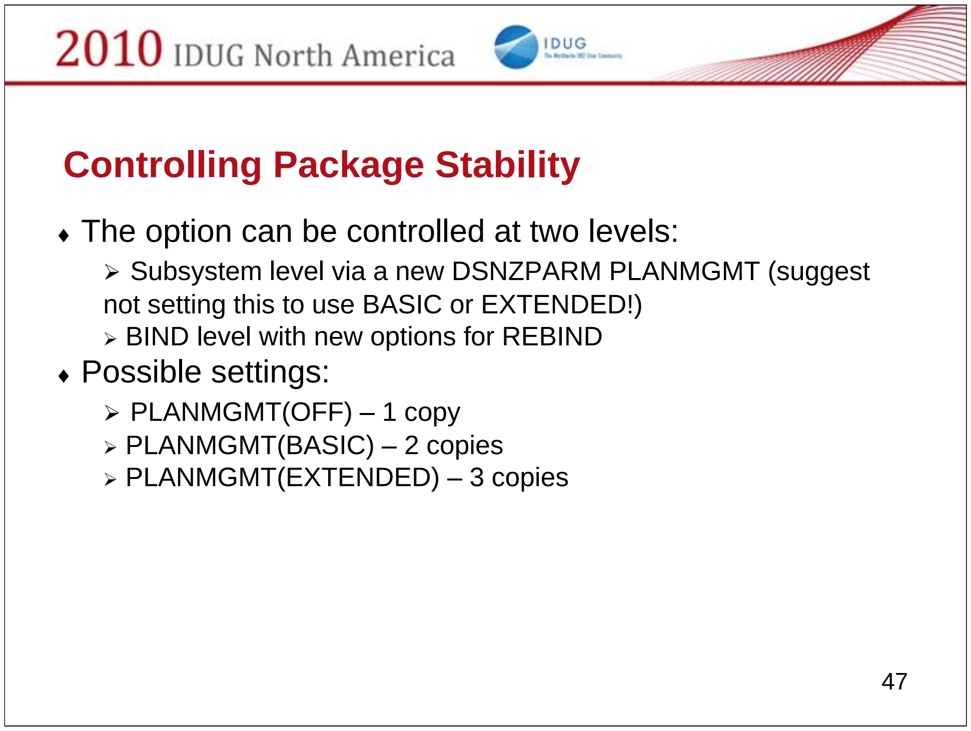
A new feature available with DB2 9.



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).

The infrastructure needed.

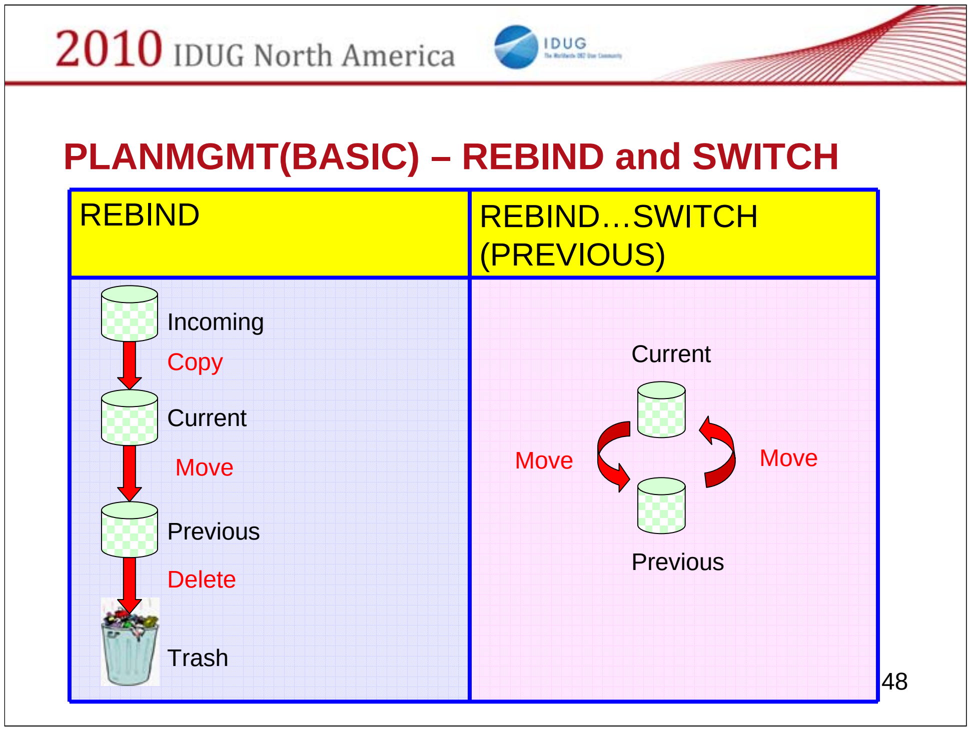

2010 IDUG North America

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

47

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



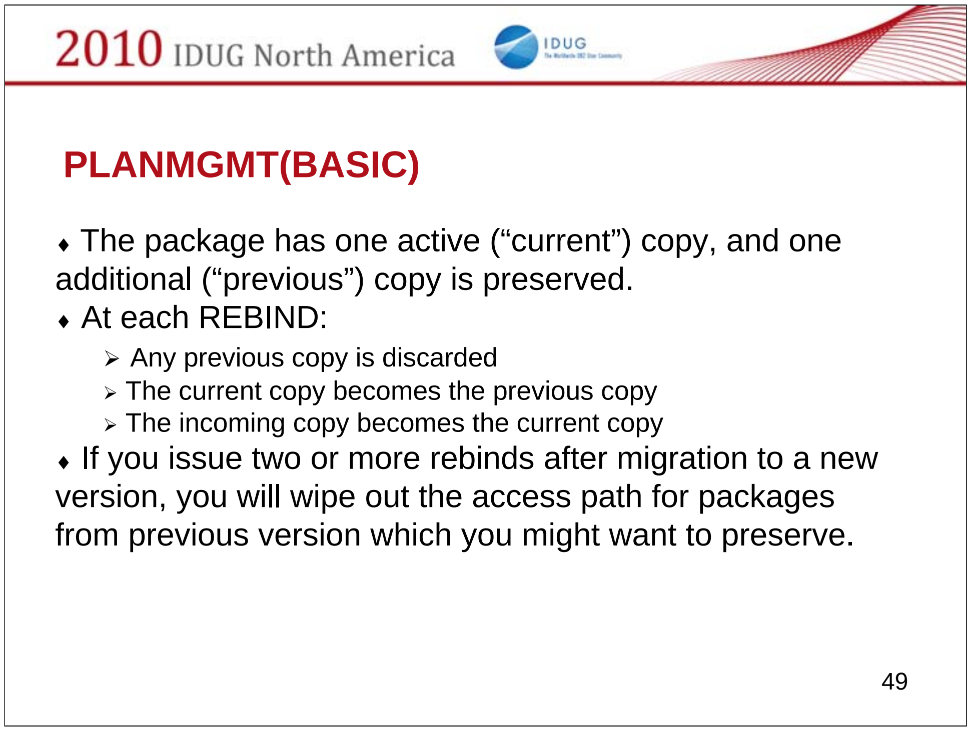
2010 IDUG North America

PLANMGMT(BASIC) – REBIND and SWITCH

REBIND	REBIND…SWITCH (PREVIOUS)

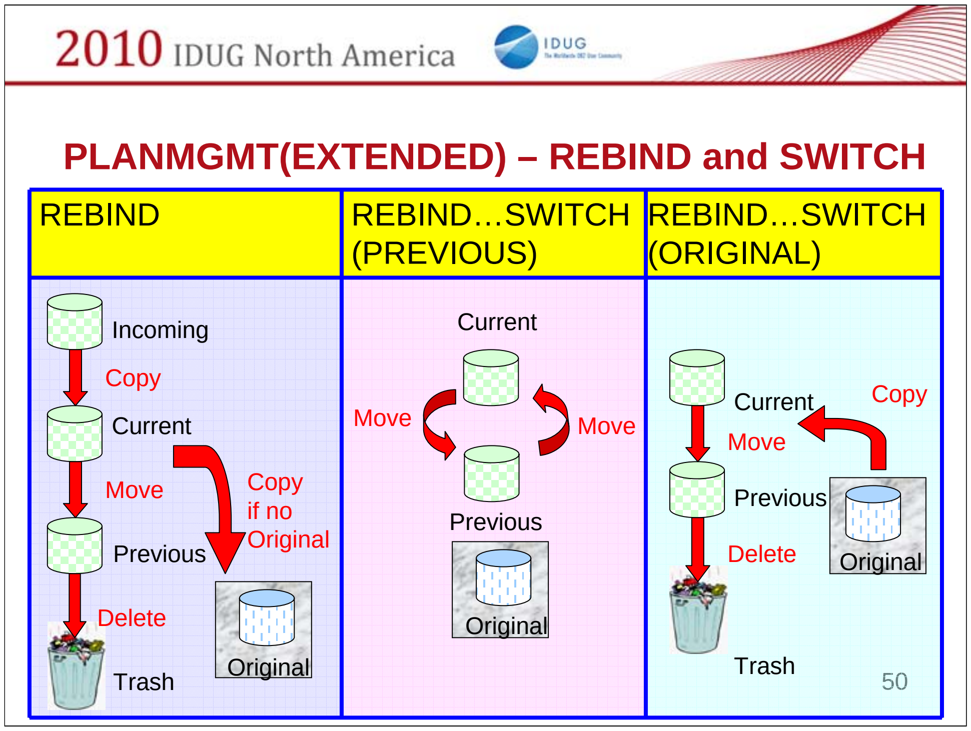
48

How does it work?



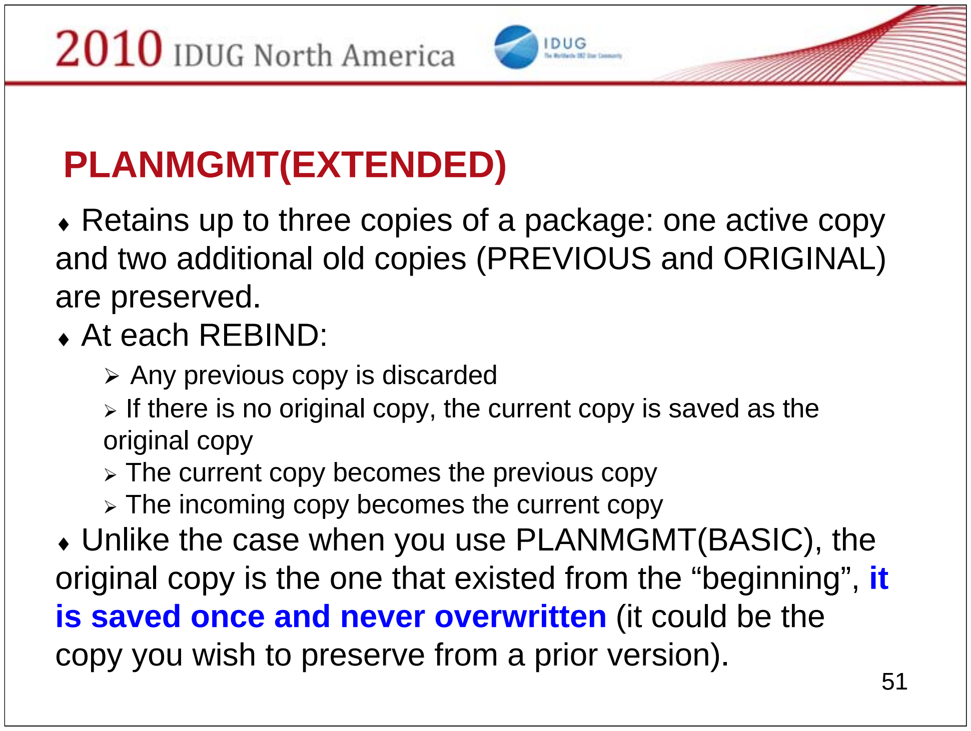
PLANMGMT(BASIC)

- 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

REBIND	REBIND…SWITCH (PREVIOUS)	REBIND…SWITCH (ORIGINAL)
Incoming Copy Current Move Copy if no Original Previous Delete Trash Original	Current Move Move Previous Original	Current Copy Move Previous Delete Original Trash

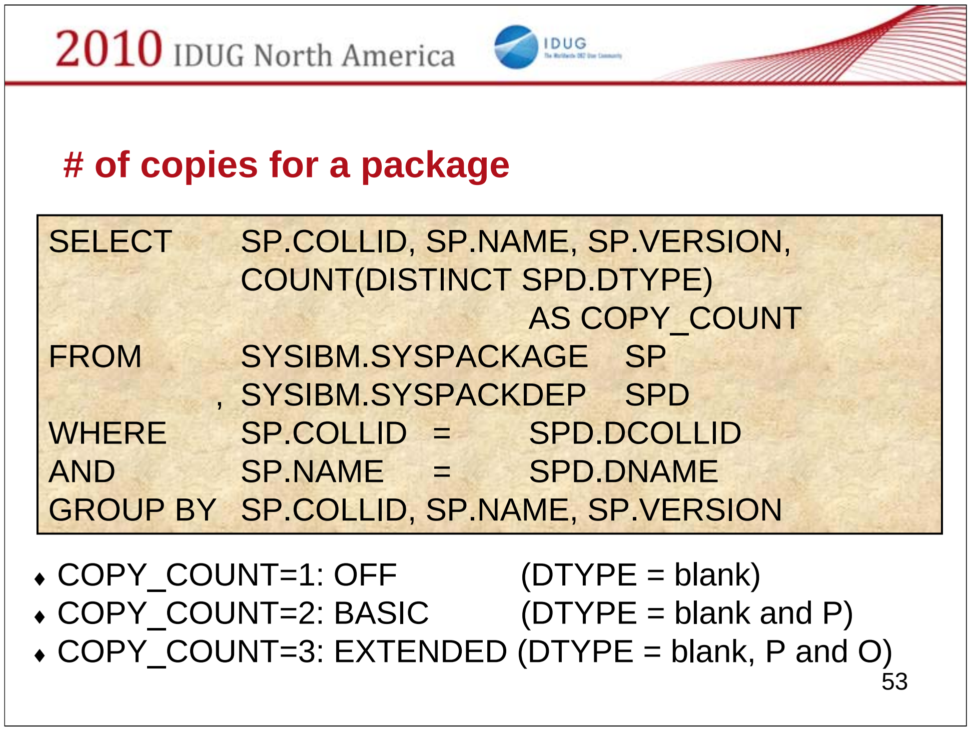


PLANMGMT(EXTENDED)

- 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.



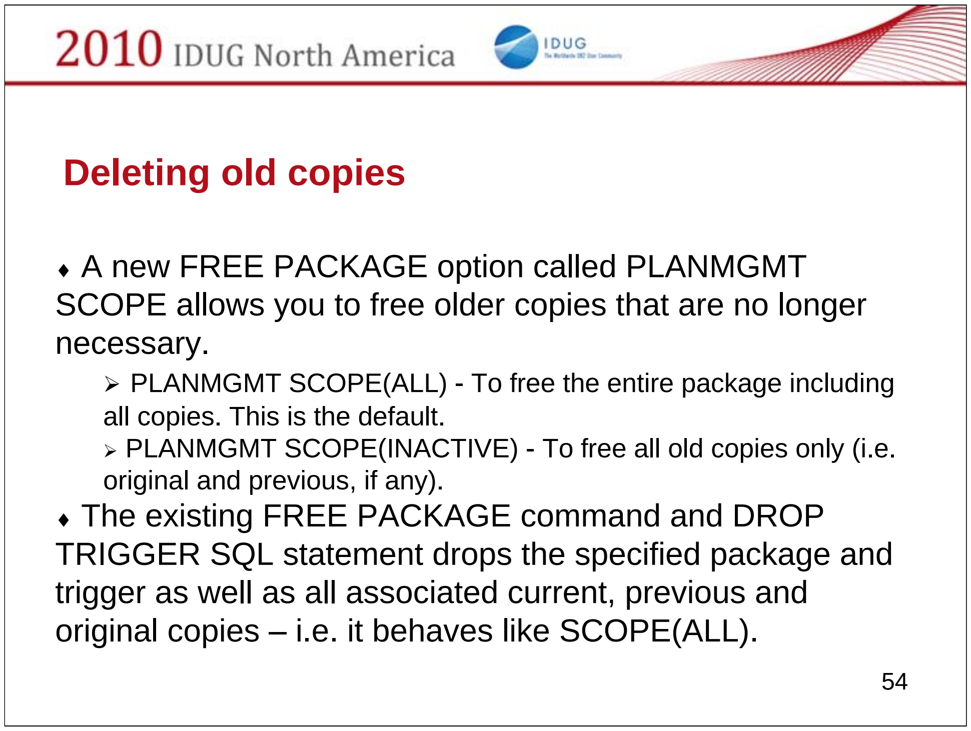
# of copies for a package

```
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)

53

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



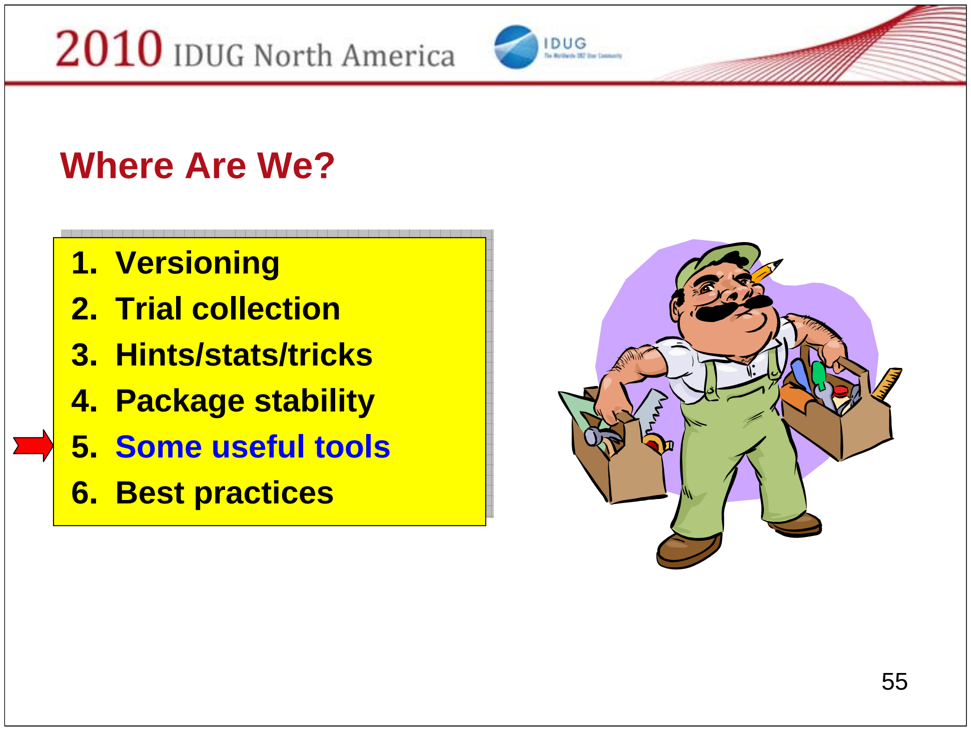
2010 IDUG North America

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).

54

Clean up of old copies.

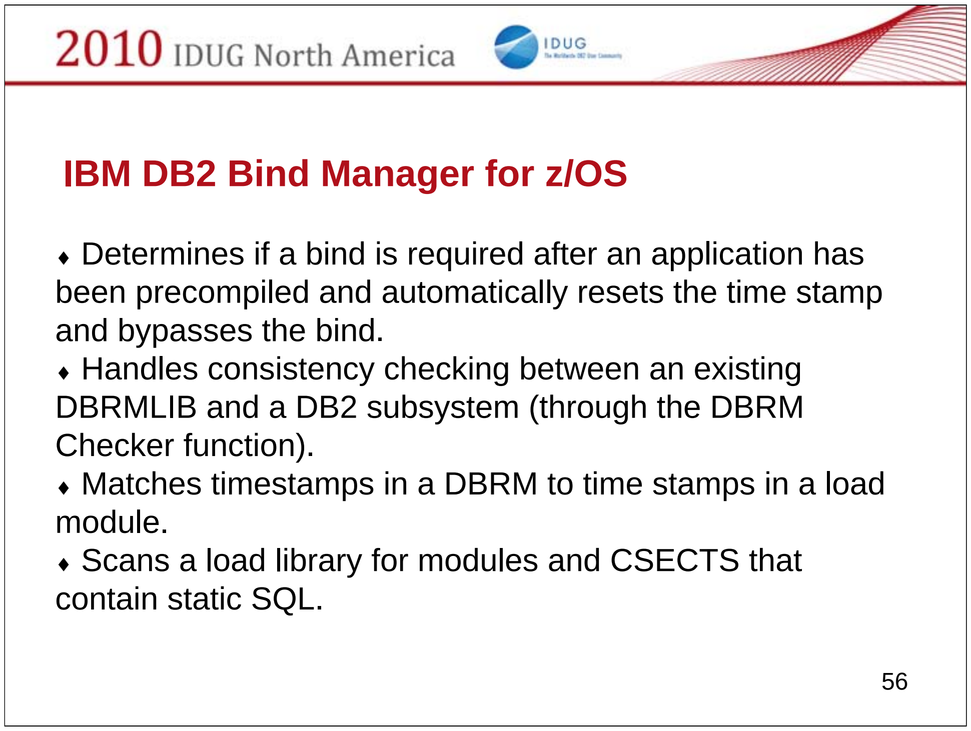


Where Are We?

1. Versioning
2. Trial collection
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. Best practices

In this section, we will briefly mention some of the available tools to make the job of access path management easier. This is not an exhaustive list nor will I get into a comparison of products.

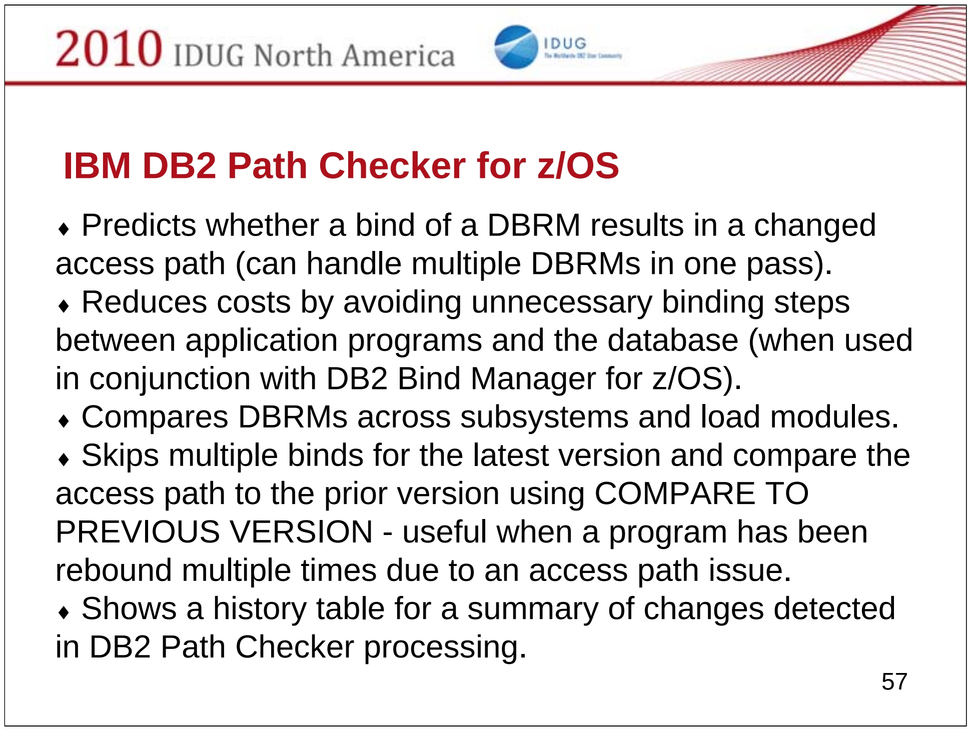
We will cover this section very quickly.



IBM DB2 Bind Manager for z/OS

- Determines if a bind is required after an application has been precompiled and automatically resets the time stamp and bypasses the bind.
- Handles consistency checking between an existing DBRMLIB and a DB2 subsystem (through the DBRM Checker function).
- Matches timestamps in a DBRM to time stamps in a load module.
- Scans a load library for modules and CSECTS that contain static SQL.

Information about Bind Manager.

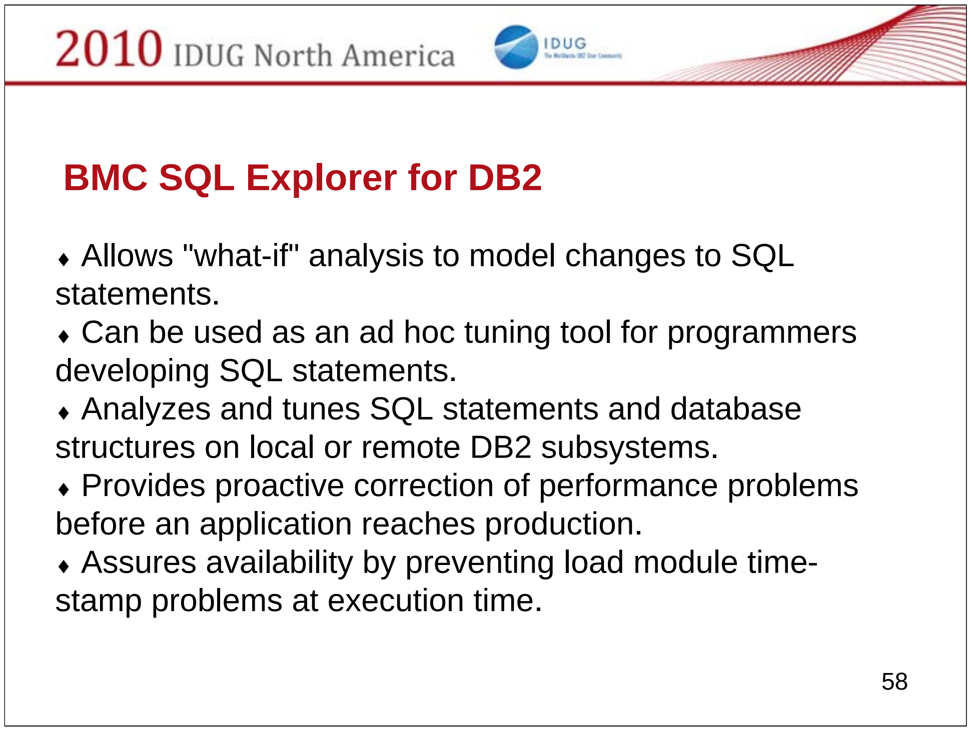

2010 IDUG North America

IBM DB2 Path Checker for z/OS

- ♦ Predicts whether a bind of a DBRM results in a changed access path (can handle multiple DBRMs in one pass).
- ♦ Reduces costs by avoiding unnecessary binding steps between application programs and the database (when used in conjunction with DB2 Bind Manager for z/OS).
- ♦ Compares DBRMs across subsystems and load modules.
- ♦ Skips multiple binds for the latest version and compare the access path to the prior version using COMPARE TO PREVIOUS VERSION - useful when a program has been rebound multiple times due to an access path issue.
- ♦ Shows a history table for a summary of changes detected in DB2 Path Checker processing.

57

Information about path Checker.



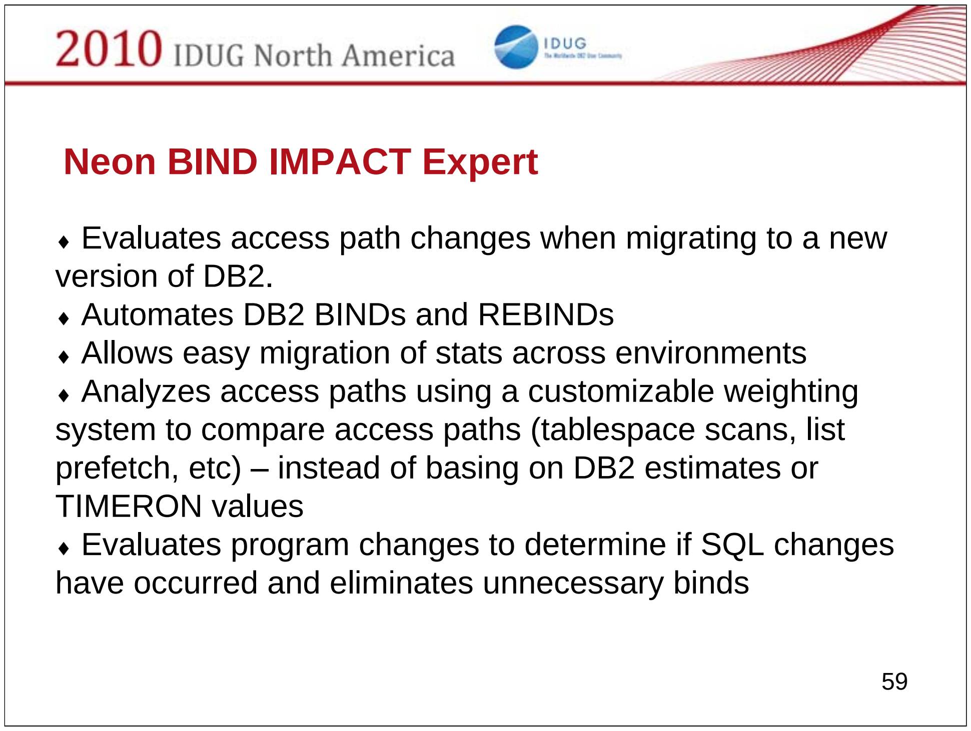
2010 IDUG North America

BMC SQL Explorer for DB2

- ♦ Allows "what-if" analysis to model changes to SQL statements.
- ♦ Can be used as an ad hoc tuning tool for programmers developing SQL statements.
- ♦ Analyzes and tunes SQL statements and database structures on local or remote DB2 subsystems.
- ♦ Provides proactive correction of performance problems before an application reaches production.
- ♦ Assures availability by preventing load module time-stamp problems at execution time.

58

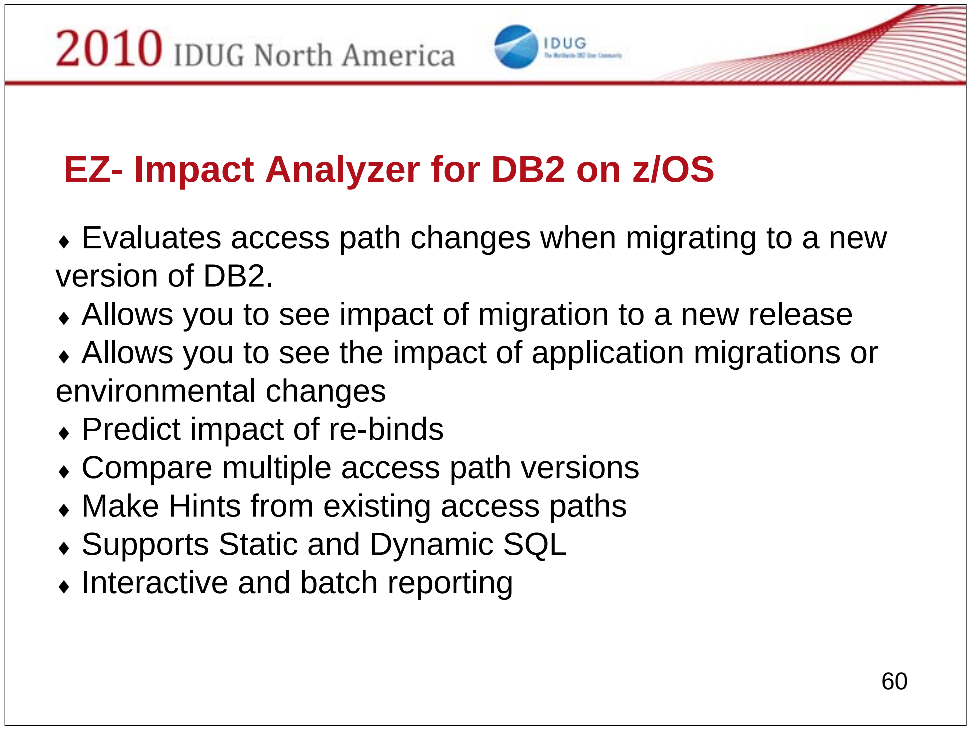
Information about BMC SQL Explorer for DB2.



Neon BIND IMPACT Expert

- Evaluates access path changes when migrating to a new version of DB2.
- Automates DB2 BINDs and REBINDs
- Allows easy migration of stats across environments
- Analyzes access paths using a customizable weighting system to compare access paths (tablespace scans, list prefetch, etc) – instead of basing on DB2 estimates or TIMERON values
- Evaluates program changes to determine if SQL changes have occurred and eliminates unnecessary binds

Information about Bind Impact Expert.



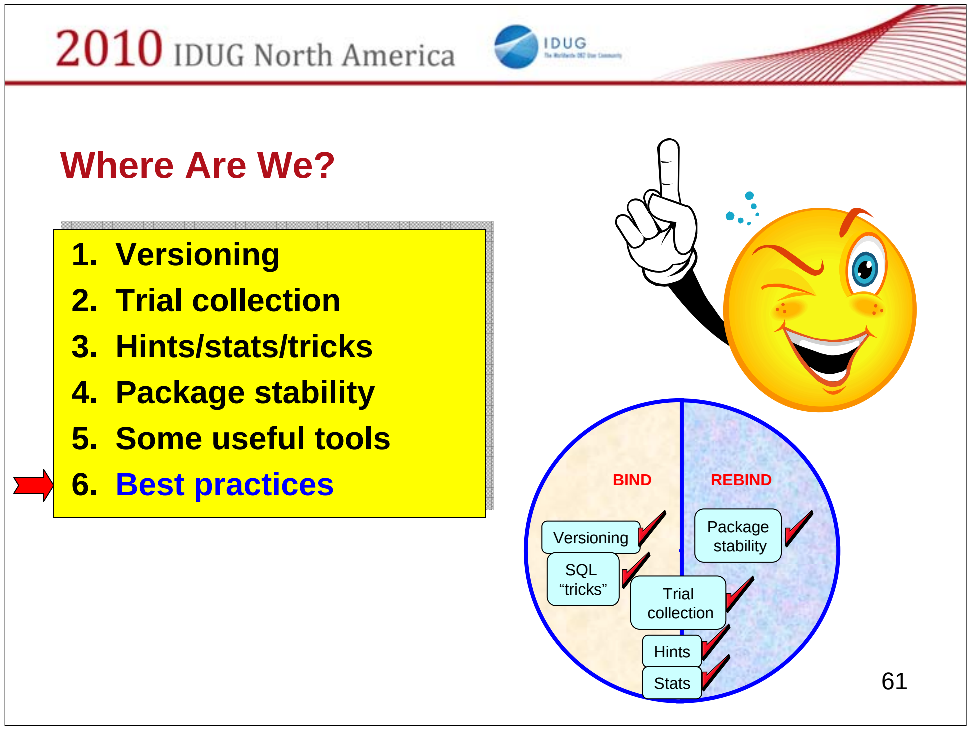
2010 IDUG North America

EZ- Impact Analyzer for DB2 on z/OS

- Evaluates access path changes when migrating to a new version of DB2.
- Allows you to see impact of migration to a new release
- Allows you to see the impact of application migrations or environmental changes
- Predict impact of re-binds
- Compare multiple access path versions
- Make Hints from existing access paths
- Supports Static and Dynamic SQL
- Interactive and batch reporting

60

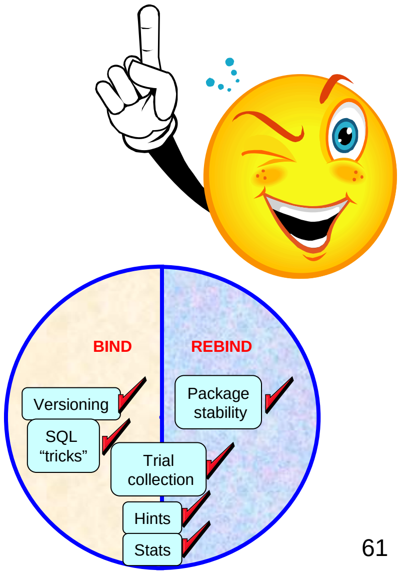
Information about EZ- Impact Analyzer for DB2 on z/OS.



2010 IDUG North America

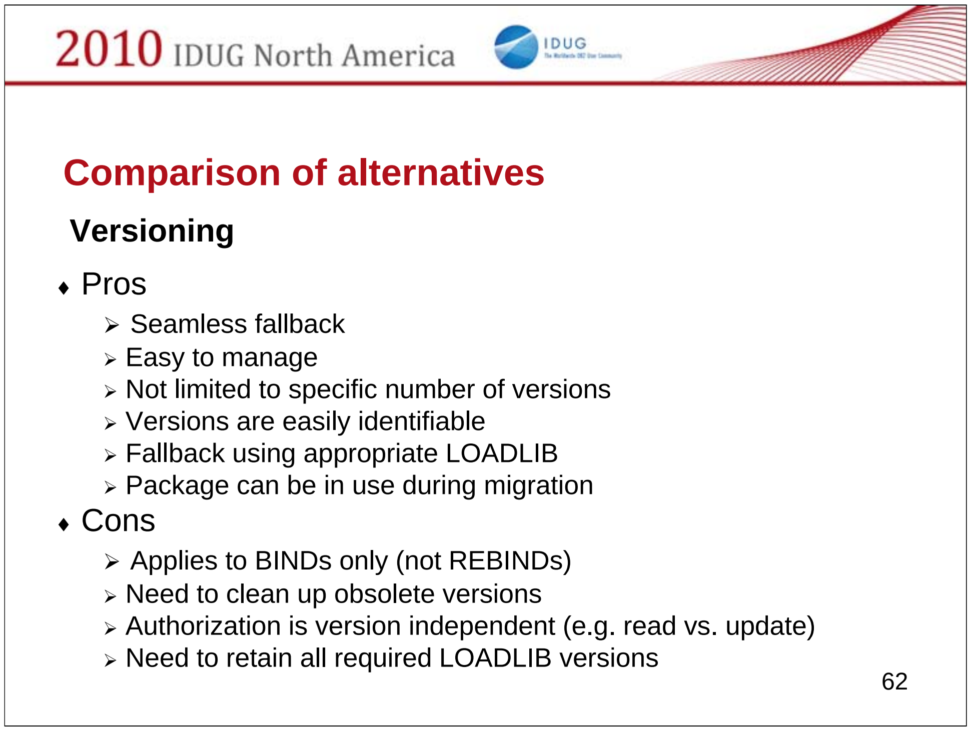
Where Are We?

1. Versioning
2. Trial collection
3. Hints/stats/tricks
4. Package stability
5. Some useful tools
6. **Best practices**



61

In this section we will compare the alternatives and recommend what I believe to be best practices.

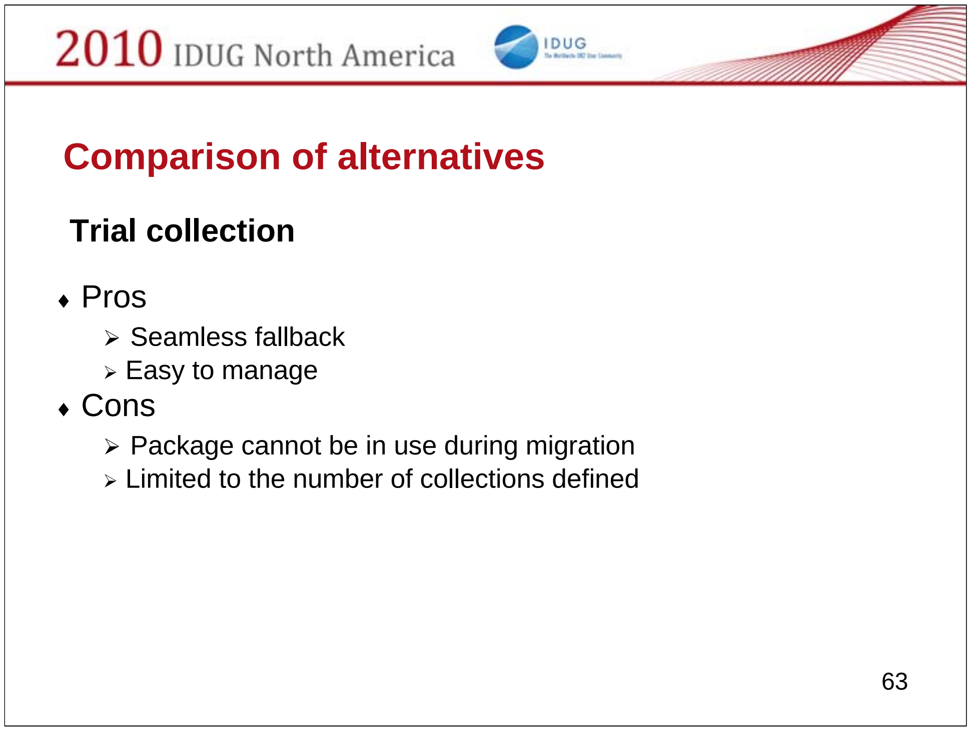


Comparison of alternatives

Versioning

- Pros
  - Seamless fallback
  - Easy to manage
  - Not limited to specific number of versions
  - Versions are easily identifiable
  - Fallback using appropriate LOADLIB
  - Package can be in use during migration
- Cons
  - Applies to BINDs only (not REBINDs)
  - Need to clean up obsolete versions
  - Authorization is version independent (e.g. read vs. update)
  - Need to retain all required LOADLIB versions

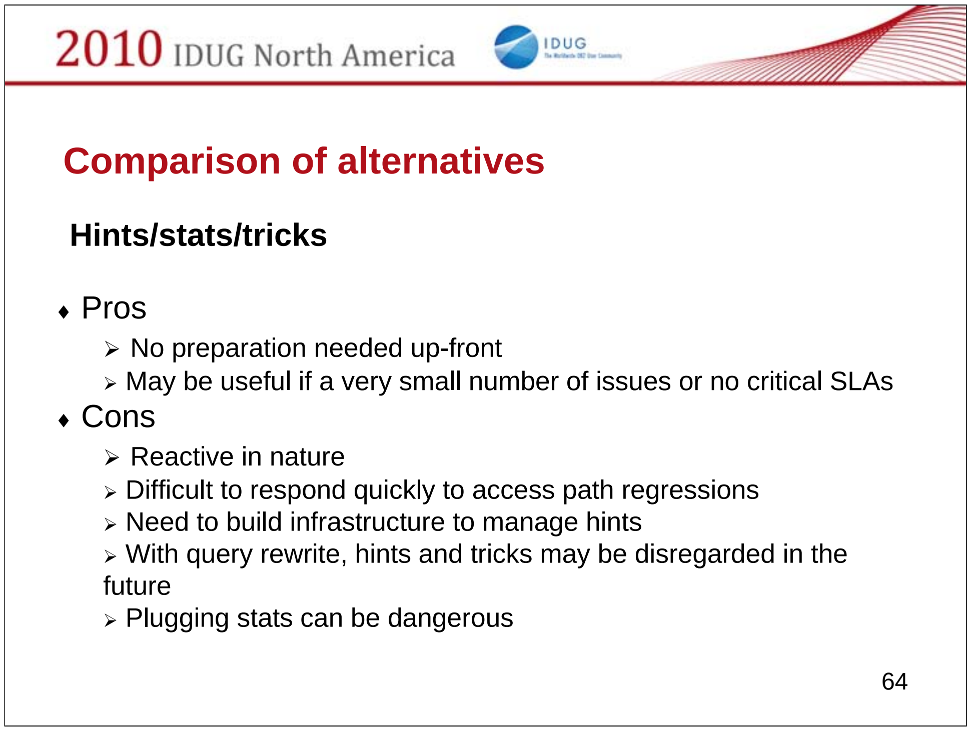
62



Comparison of alternatives

Trial collection

- Pros
  - Seamless fallback
  - Easy to manage
- Cons
  - Package cannot be in use during migration
  - Limited to the number of collections defined


2010 IDUG North America

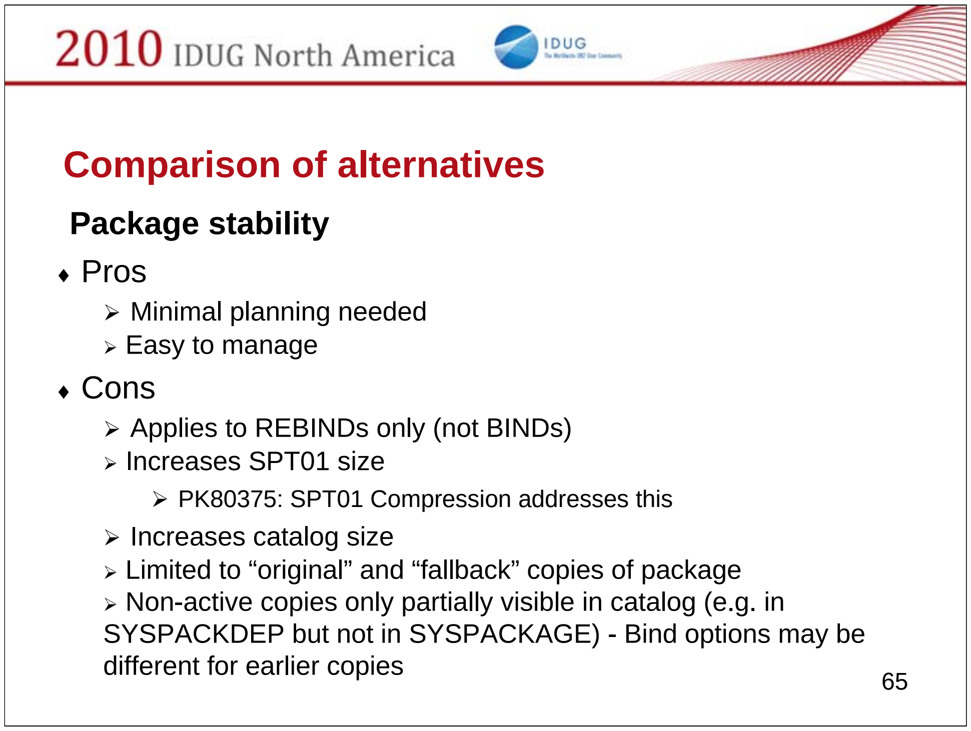
Comparison of alternatives

Hints/stats/tricks

- Pros
  - No preparation needed up-front
  - May be useful if a very small number of issues or no critical SLAs
- Cons
  - Reactive in nature
  - Difficult to respond quickly to access path regressions
  - Need to build infrastructure to manage hints
  - With query rewrite, hints and tricks may be disregarded in the future
  - Plugging stats can be dangerous

64

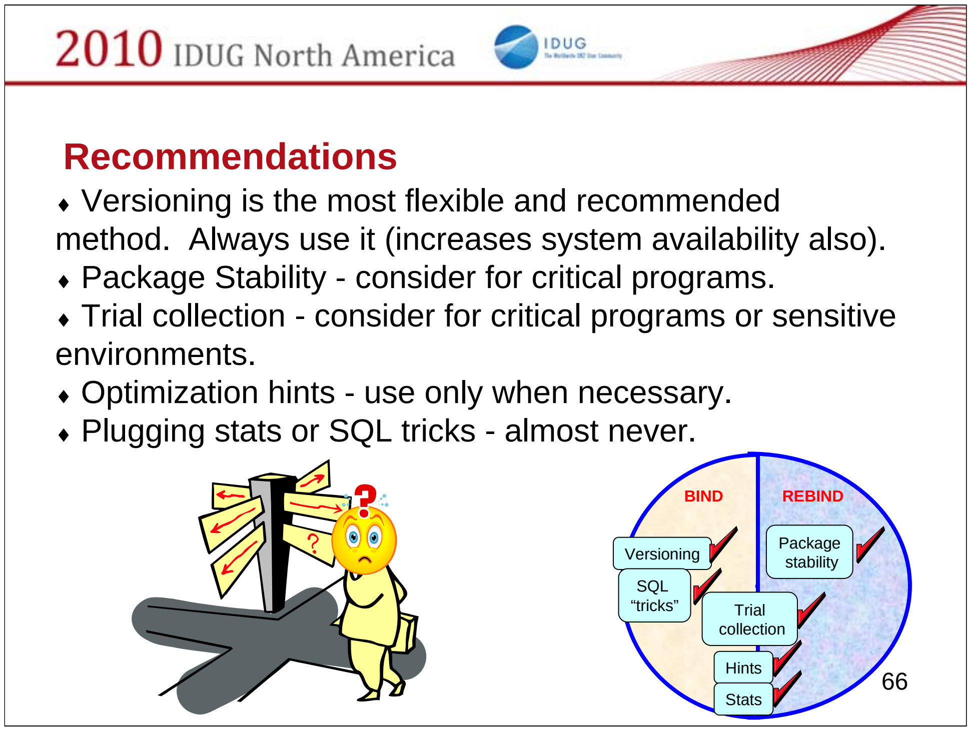
My biggest issue with hints is that I need “anti-hints” – I don’t know what works necessarily, but do know what does NOT work.



Comparison of alternatives

Package stability

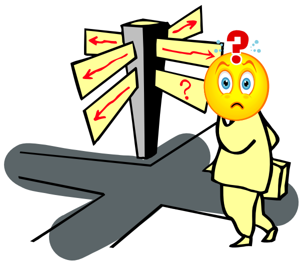
- Pros
  - Minimal planning needed
  - Easy to manage
- Cons
  - Applies to REBINDs only (not BINDs)
  - Increases SPT01 size
    - PK80375: SPT01 Compression addresses this
  - Increases catalog size
  - Limited to “original” and “fallback” copies of package
  - Non-active copies only partially visible in catalog (e.g. in SYSPACKDEP but not in SYSPACKAGE) - Bind options may be different for earlier copies

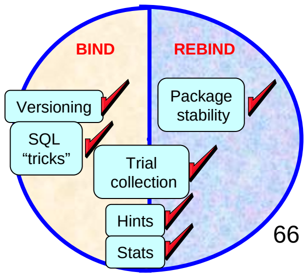


2010 IDUG North America

Recommendations

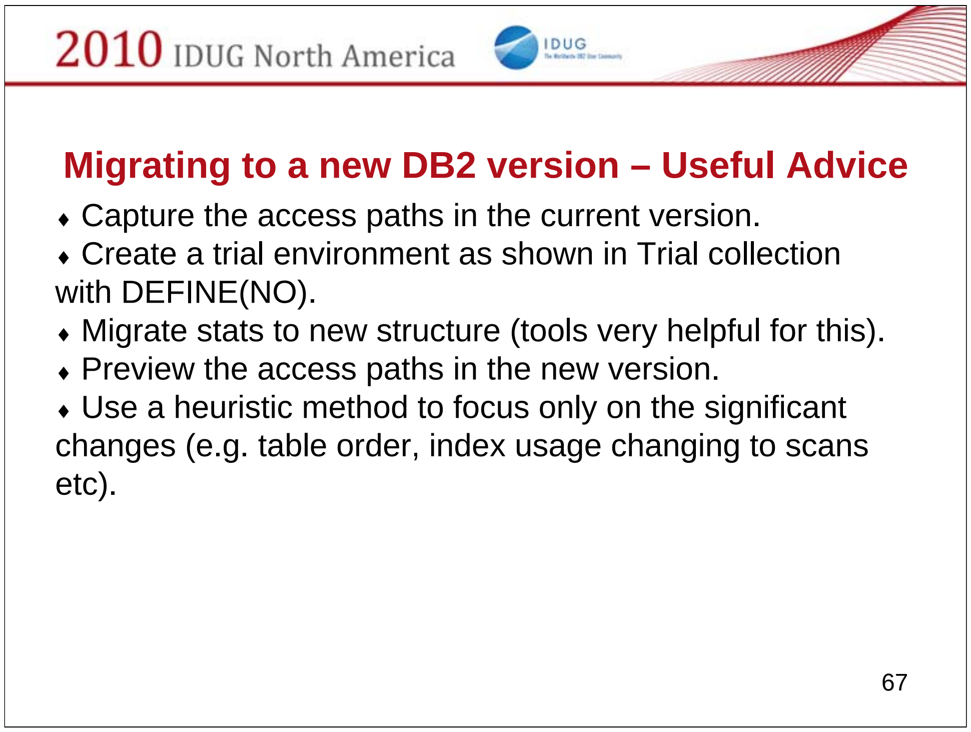
- Versioning is the most flexible and recommended method. Always use it (increases system availability also).
- Package Stability - consider for critical programs.
- Trial collection - consider for critical programs or sensitive environments.
- Optimization hints - use only when necessary.
- Plugging stats or SQL tricks - almost never.





66

A summary of the recommendations.

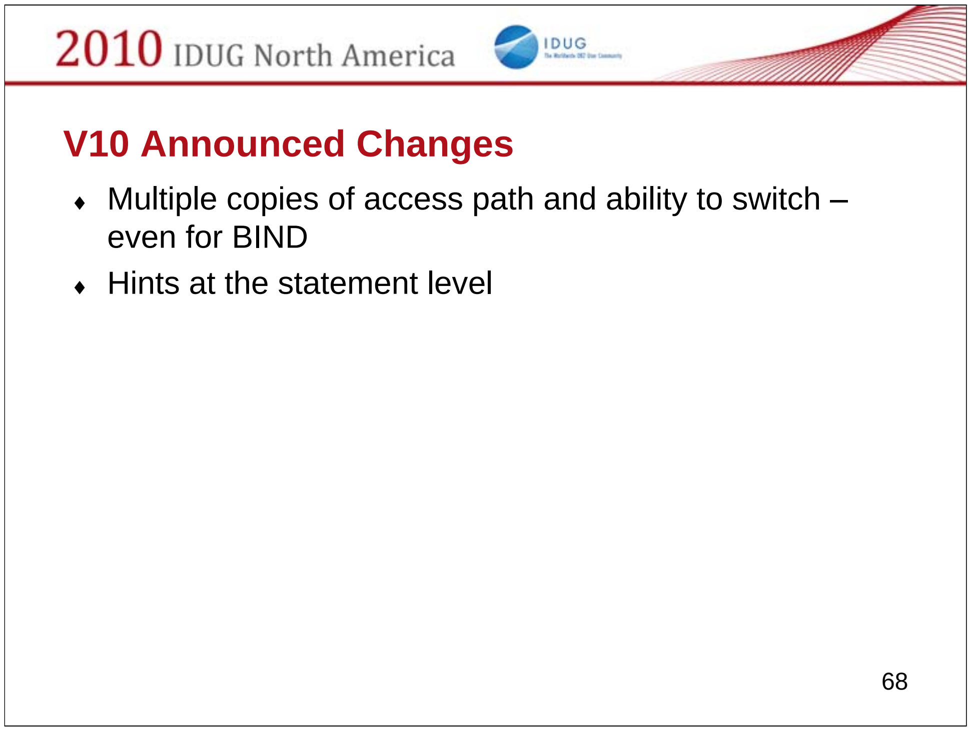

2010 IDUG North America

Migrating to a new DB2 version – Useful Advice

♦ Capture the access paths in the current version.
♦ Create a trial environment as shown in Trial collection with DEFINE(NO).
♦ Migrate stats to new structure (tools very helpful for this).
♦ Preview the access paths in the new version.
♦ Use a heuristic method to focus only on the significant changes (e.g. table order, index usage changing to scans etc).

67

This is the suggested due diligence to make sure there are no surprises.


2010 IDUG North America

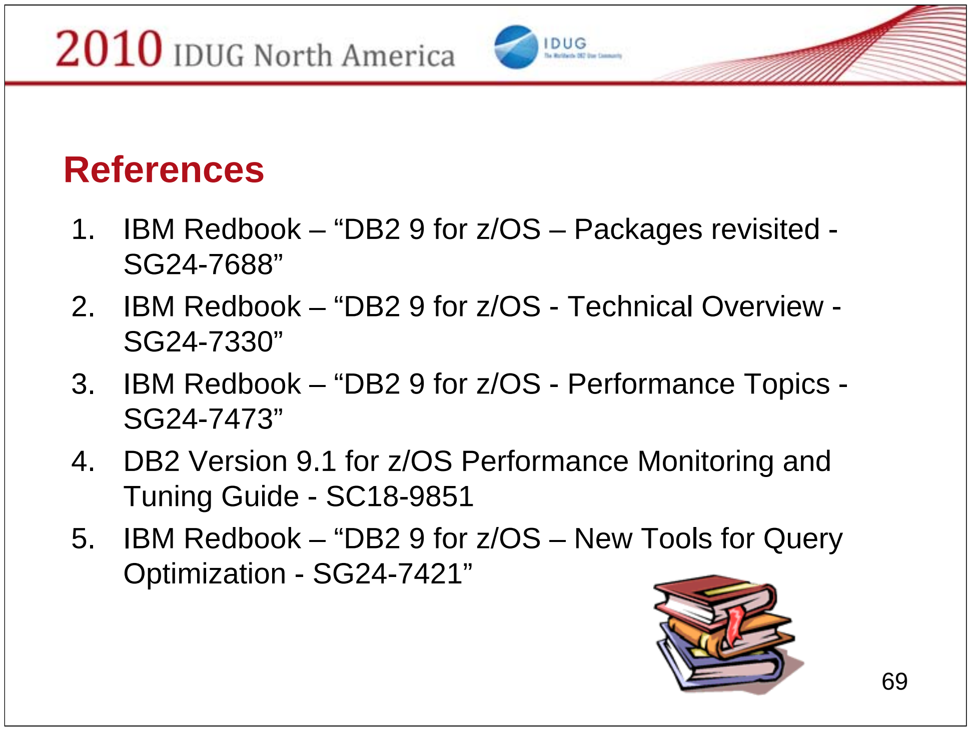
V10 Announced Changes

- Multiple copies of access path and ability to switch – even for BIND
- Hints at the statement level

68

Some of the many enhancements which will be covered in other IBM sessions.

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


2010 IDUG North America

References

1. IBM Redbook – “DB2 9 for z/OS – Packages revisited - SG24-7688”
2. IBM 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. IBM Redbook – “DB2 9 for z/OS – New Tools for Query Optimization - SG24-7421”

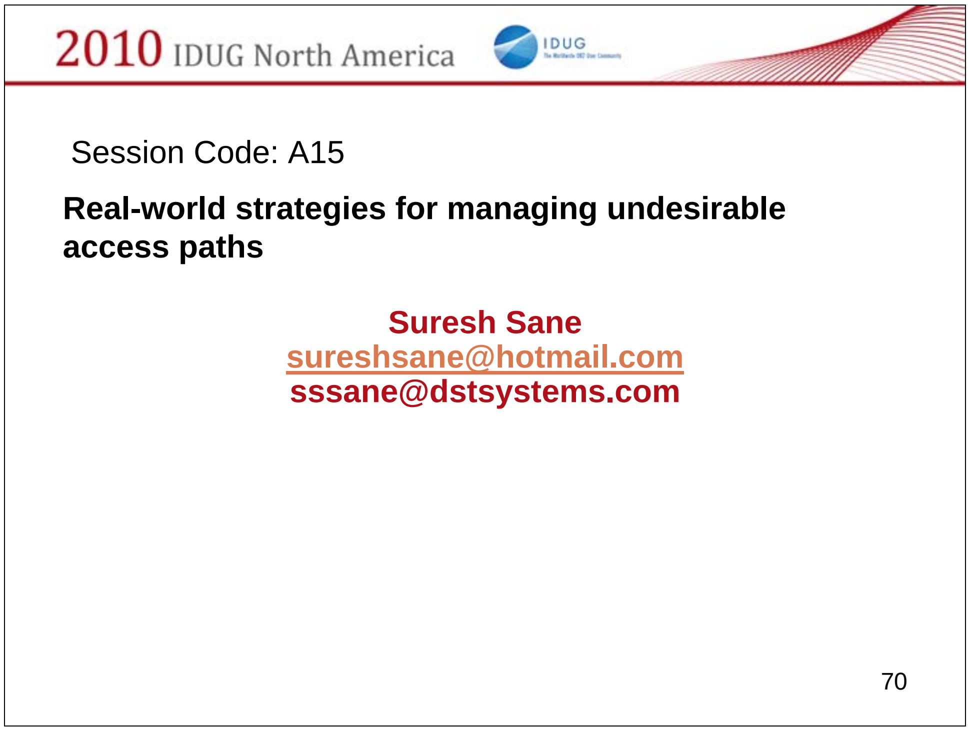
69

Some of the many useful references.

Another useful reference is the IBM Redbook “UDB for z/OS: Application Design for High Performance and Availability” – SG24-7134 which provides excellent tips on how to obtain a desirable access path in the first place.

ACKNOWLEDGEMENTS:

I gave an early version of this at the Central Canada DB2 User Group meeting in June 2009. Following this, I received some excellent feedback. I would like to express my sincere thanks to Rick Butler (BMO Financial Group, Toronto ON) for his invaluable help in revising the material. Thank you Rick!



2010 IDUG North America

Session Code: A15

Real-world strategies for managing undesirable access paths

Suresh Sane
sureshsane@hotmail.com
sssane@dstsystems.com

70

I trust this session has empowered you with the knowledge manage your access paths effectively. Hope your path is always the right one!

Thank you!