My Favourite Db2 Query Optimization Registry Variables

John Hornibrook

IBM Canada

Platform: Db2/LUW

Optimal query access plans are essential for good data server performance and it is the query optimizer's job to choose the best one. For most systems, only minimal tuning is necessary if automatic statistics collection is enabled. However, there are more complex scenarios and configurations where the query optimizer needs some help to get the best access plan and there are Db2 registry variables available to handle these. Query optimization registry variables are dynamic so they can be applied without restarting the database and they can also be specified using optimization guidelines. Attend this session to learn what is available and how they can help improve query performance for your database.

Agenda

- Learn about a number of query optimization registry variables, how they work and when to use them.
- Understand the available options for specifying query optimization registry variables.
- Gain some insight into query optimization.

What are Query Optimization Registry Variables?

- The same as regular registry variables, but they only affect query optimization
  - Query transformations
  - Access plan generation
  - Query optimization algorithms
- Subset of SQL Compiler registry variables
  - Only affect SQL compilation i.e. not runtime
  - Have some special features:
    - Dynamic
    - Can be set using optimization guidelines (aka hints)

3

Why is this list my favourites?

- They allows Db2 users to resolve query performance issues without guidance from IBM support/service
- They aren't too complex or esoteric
- Some aren't documented
  - Unless IDUG presentations count as documentation?

- Caveat: all of them have tradeoffs
  - They might not work well for all queries
  - Performance of some queries could regress
  - They might work well for applications with certain characteristics
    - E.g. OLAP vs OLTP
- Test thoroughly before deploying in production!

SQL Compiler Registry Variables

- SQL compiler registry variables only affect SQL compilation
  - i.e. not Db2 runtime
- They are dynamic
  - i.e. no instance restart is necessary (`db2set -im`)
- They can be specified using optimization guidelines

```
<OPTGUIDELINES>
      <REGISTRY>
         <OPTION NAME='DB2_SELECTIVITY' VALUE='YES'/>
         <OPTION NAME='DB2_REDUCED_OPTIMIZATION' VALUE='YES'/>
      </REGISTRY>
 </OPTGUIDELINES>
```

- More on this later …

5

SQL compiler registry variables affect only SQL compilation i.e. not Db2 runtime. They are also dynamic so they can be changed without an instance restart. Note that cached dynamic SQL will not be recompiled when db2set –im is specified.
SQL compiler registry variables can be specified in optimization guidelines and profiles. Optimization profiles can have different registry variable values applied to a specific query statement or to many query statements used in an application.

Setting registry variables in an optimization profile can increase the flexibility you have in using different query statements for different applications. When you use the db2set command to set registry variables, the registry variable values are applied to the entire instance. In optimization profiles, the registry variable values apply only to sessions that have the optimization profile active. The registry variables can be further limited to certain statements specified in the optimization profile. By setting registry variables in an optimization profile, you can tailor specific statements for applications without worrying about the registry variable settings of other query statements.
Only a subset of registry variables can be set in an optimization profile.
See here for the full list:
https://www.ibm.com/docs/en/db2/11.5?topic=profiles-sql-compiler-registry-variables-in-optimization-profile

DB2_OPTIMIZER_VERSION (1|3) 11.5.4

- Specifies the query optimizer version level
- Setting it to a previous version level disables query optimizations introduced after that level
- Does not control SQL syntax or semantics
- Version level must be specified in the form: VV.RR.MM.FF
  - VV=version, RR=release, MM=modification level, FF=fix pack level
  - E.g. DB2_OPTIMIZER_VERSION=11.1.0.0
    - (aka 11.5 GA)
- Earliest supported optimizer version is 10.5.0.0
  - Modification level is always 0 for Db2 10.5

DB2_OPTIMIZER_VERSION (2|3)

- Controls optimizer behavior that is currently controlled by existing registry variables
  - Since 10.5, all significant optimizer changes are controlled by registry variables
  - Most are undocumented
- Registry variables controlled by DB2_OPTIMIZER_VERSION can be specified at the same time and will take precedence e.g.

```
db2set DB2_OPTIMIZER_VERSION=11.1.0.0
   (implicitly sets DB2_REDUCED_OPTIMIZATION=XYZ)
db2set DB2_REDUCED_OPTIMIZATION=ABC
   (DB2_REDUCED_OPTIMIZATION=ABC takes precedence)
```

- This preserves the optimizer behaviour in the previous release, if other SQL compiler registry variables were set

7

DB2_OPTIMIZER_VERSION (3|3)

- Intended to be used for emergency purposes
- Typical scenario is after a Db2 upgrade if performance regresses due to access plan changes
- If this reg var helps, work with Db2 service to determine which optimizer change was the issue
  - The particular optimizer change can be disabled by a specific registry variable
  - If DB2_OPTIMIZER_VERSION is left set to a previous release, you will not realize the benefit of optimizer changes in the new release
- Sometimes additional statistics will avoid the need to disable optimizer features/fixes

DB2_SELECTIVITY

- Controls where the SELECTIVITY clause can be used
  - Documented options: YES, NO, or ALL
  - Default=NO (ALL when DB2_WORKLOAD=SAP since 11.5.5)
- But first - what is ***selectivity*** ?
- The optimizer estimates the amount of filtering done by each search condition (predicate)
  - Also known as ***filter factor***
  - A value between 0 and 1 that represents the percentage of rows that satisfy the predicate
  - e.g. 0.03 means that 3% of the rows qualify
- Selectivity is used to determine the number of rows processed by each access plan operator
  - ***cardinality***
  - e.g. 10000 rows * 0.03 = 300
- The cardinality is critical for estimating the cost of an access plan operator
- These estimates are shown in the explain output

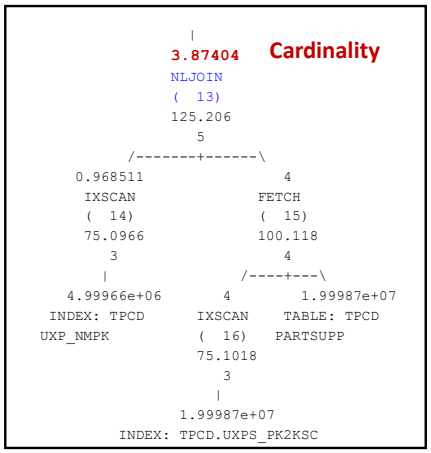
Selectivity and Cardinality

Selectivity (filter factor) and cardinality as seen in db2exfmt

```
13) NLJOIN: (Nested Loop Join)

 Predicates:
 ----------
 16) Predicate used in Join
     Relational Operator:     Equal (=)
     Subquery Input Required: No
     Filter Factor:           5.00034e-08

     Predicate Text:
     --------------
    (Q1.PS_PARTKEY = Q2.P_PARTKEY)
```



Achieving Accurate Selectivity Estimates

- Everybody knows the best way to get accurate predicate selectivity estimate, right?
- **RUNSTATS of course!**
  - Distribution statistics
  - Column group statistics
  - Statistical views
  - Enable automatic statistics collection
- But sometimes the optimizer still gets it wrong, even with the best statistics …

SELECTIVITY clause

- There is another option – the **SELECTIVITY** clause

```
SELECT NAME FROM EMPLOYEE WHERE EMP_ID = ? SELECTIVITY 0.003
```

- Requires setting registry variable DB2_SELECTIVITY=ON
- Supported for basic predicates with input variables
  - =, >, <, !=
  - Not supported for LIKE, BETWEEN, subquery predicates, OR, NOT
  - BETWEEN can be manually rewritten as >=, <=
  - SELECTIVITY should be specified for each OR subterm

```
C1 >= ? SELECTIVITY 0.25 OR
C2 <= ? SELECTIVITY 0.125
```

  - Subquery predicates could be rewritten to joins

12

SELECTIVITY clause

- The ultimate option: DB2_SELECTIVITY=ALL
- Allows use with predicates with literals
- Allows use with LIKE predicates
  - Except with trailing 'match all' (%)
  - `C1 LIKE 'ABC%'` is rewritten to `C1 BETWEEN 'ABC<low chars>' AND 'ABC<high chars>'`
  - SELECTIVITY not supported for BETWEEN
- Behavior might be unpredictable with more complex predicates because they are transformed e.g. subquery predicates

13

SELECTIVITY clause – cautions

- SELECTIVITY might not be practical to use if the selectivity can vary significantly for different input variable values or literals
- The selectivity value might need to be updated periodically as the data changes
- It could still be better than the optimizer's estimate, which could be a default value
  - Typically 0.04, but can vary depending on the column's cardinality
- Consider using the REOPT option for queries with input variables
  - The optimizer is missing information needed to compute accurate selectivity estimates when input variables are used
  - Can't compare input value to column statistics
  - Can't recognize skew

14

REOPT bind option

- REOPT bind option indicates to defer optimization until input variables are available
  - **REOPT (ALWAYS)** – re-optimize statement upon each execution (no dynamic statement caching)
  - **REOPT(ONCE)** – cache plan based on first set of input variables obtained
- REOPT(ALWAYS)
  - Good choice for **complex queries**
- REOPT(ONCE)
  - Sometimes a good choice for **OLTP**
  - If the first input values:
    - provides a better access plan than optimizer's defaults
    - are representative of subsequent input variables
- See Db2 documention for how to set for various application types
- https://www.ibm.com/docs/en/db2/10.5?topic=wtqop-using-reopt-bind-option-input-variables-in-complex-queries

15

There are a number of ways that REOPT can be specified:
For embedded SQL in C/C++ applications, use the REOPT BIND option. This BIND option affects re-optimization behavior for both static and dynamic SQL.
For CLP packages, rebind the CLP package with the REOPT bind option. For example, to rebind the CLP package used for isolation level CS with REOPT ALWAYS, specify the following command:

rebind nullid.SQLC2G13 reopt always;

For CLI applications or JDBC applications using the legacy JDBC driver, use the REOPT keyword setting in the db2cli.ini configuration file. The values and their options are:

2 - NONE
3 - ONCE
4 - ALWAYS

For JDBC applications using the JCC Universal Driver, use one of the following approaches:

Use the SQL_ATTR_CURRENT_PACKAGE_SET connection or statement attribute to specify either the NULLID, NULLIDR1 or NULLIDRA package sets. The NULLIDR1 and NULLIDRA are reserved package set names. When used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with the following commands:
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA

For SQL PL procedures, use one of the following approaches:

Use the SET_ROUTINE_OPTS stored procedure to set the bind options to be used for the creation of SQL PL procedures within the current session. For example, call sysproc.set_routine_opts('reopt always');

Use the **DB2_SQLROUTINE_PREPOPTS** registry variable to set the SQL PL procedure options at the instance level. Values set using the SET_ROUTINE_OPTS stored procedure will override those specified with **DB2_SQLROUTINE_PREPOPTS**

SELECTIVITY clause – some history

- Originally limited to user-defined predicates

```
SELECT *
     FROM STORES
     WHERE myfunction(parm,parm) = 1 SELECTIVITY 0.004
```

- SELECTIVITY is not part of the SQL standard for built-in predicates
- But it was determined to be very useful, so the restriction was gradually lifted
- DB2_SELECTIVITY might not be needed in future releases

16

DB2_REDUCED_OPTIMIZATION

- Original purpose:
  - Disable certain query optimization algorithms in order to **reduce prepare time**
- Has been extended to control many optimization algorithms, with the goal to avoid slow access plans
- Documented options:
  - NO, YES, any integer, DISABLE, JUMPSCAN, NO_SORT_NLJOIN, NO_SORT_MGJOIN, ZZJN, ZZJN_MULTI_FACT
  - Default=NO
- Only some of these are my favourites

17

https://www.ibm.com/docs/en/db2/11.5?topic=performance-setting-db2-reduced-optimization-registry-variable
If setting the optimization class does not reduce the compilation time sufficiently for your application, try setting the DB2_REDUCED_OPTIMIZATION registry variable. This registry variable provides more control over the optimizer's search space than setting the optimization class. This registry variable lets you request either reduced optimization features or rigid use of optimization features at the specified optimization class. If you reduce the number of optimization techniques used, you also reduce time and resource use during optimization.
Although optimization time and resource use might be reduced, there is increased risk of producing a less than optimal query access plan.

If the YES setting does not provide a sufficient reduction in compilation time, try setting the registry variable to an integer value. The effect is the same as YES, with the following additional behavior for dynamically prepared queries optimized at class 5. If the total number of joins in any query block exceeds the setting, the optimizer switches to greedy join enumeration instead of disabling additional optimization techniques. The result is that the query will be optimized at a level that is similar to optimization class 2.

DB2_REDUCED_OPTIMIZATION - YES

- 'YES' effect depends on the query optimization level
  - A value between 0 and 9
  - Can be specified
    - DFT_QUERYOPT DB config parm
    - CURRENT QUERY OPTIMIZATION special register
    - QUERYOPT bind parameter
    - QRYOPT optimization guideline
- Determines the *join enumeration method* (among other things)
- Join enumeration can be the most expensive optimization algorithm, depending on SQL statement complexity
  - Greedy – fast, but not exhaustive, could miss some good plans
  - Dynamic – exhaustive, but can take a long time for many joins

18

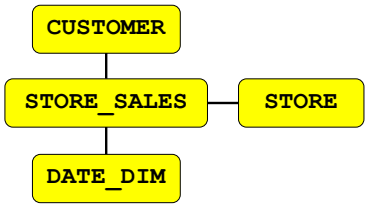
Greedy Join Enumeration

Only the cheapest join partition from each stage moves to the next stage

{ STORE_SALES (Q4) }, { CUSTOMER (Q1) }
{ STORE_SALES (Q4) }, { STORE (Q2) }
{ STORE_SALES (Q4) }, { DATE_DIM (Q3) }

{ CUSTOMER (Q1) }, **{ STORE (Q2), STORE_SALES (Q4) }**
{ DATE_DIM (Q3) }, { STORE (Q2), STORE_SALES (Q4) }

{ CUSTOMER (Q1) }, { STORE (Q2), DATE_DIM (Q3), STORE_SALES (Q4) }

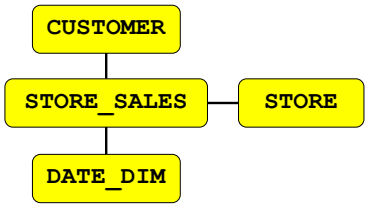


19

Dynamic Join Enumeration

{ CUSTOMER (Q1) }, { STORE_SALES (Q4) }
{ STORE (Q2) }, { STORE_SALES (Q4) }
{ DATE_DIM (Q3) }, { STORE_SALES (Q4) }

{ CUSTOMER (Q1) }, { DATE_DIM (Q3), STORE_SALES (Q4) }	P4
{ CUSTOMER (Q1) }, { STORE (Q2), STORE_SALES (Q4) }	P5
{ STORE (Q2) }, { DATE_DIM (Q3), STORE_SALES (Q4) }	P6
{ STORE (Q2) }, { CUSTOMER (Q1), STORE_SALES (Q4) }	P5
{ DATE_DIM (Q3) }, { STORE (Q2), STORE_SALES (Q4) }	P6
{ DATE_DIM (Q3) }, { CUSTOMER (Q1), STORE_SALES (Q4) }	P4



{ CUSTOMER (Q1) }, { STORE (Q2), DATE_DIM (Q3), STORE_SALES (Q4) }
{ STORE (Q2) }, { CUSTOMER (Q1), DATE_DIM (Q3), STORE_SALES (Q4) }
{ DATE_DIM (Q3) }, { CUSTOMER (Q1), STORE (Q2), STORE_SALES (Q4) }

Query Optimization Levels

- **Greedy join enumeration**
  - 0 - minimal optimization for OLTP
  - 1 - low optimization, no HSJOIN, IXSCAN, limited query rewrites
  - 2 - full optimization, limit space/time
    - use same query transforms & join strategies as class 5
- **Dynamic join enumeration**
  - 3 - moderate optimization, more limited plan space
  - 5 - self-adjusting full optimization (default)
    - uses all techniques with heuristics
    - falls back to Greedy in extreme cases
      - complex query AND (bufferpool < 2000 pages OR CPUSPEED > 3.5e-5)
  - 7 - full optimization
    - similar to 5, without heuristics
  - 9 - maximal optimization
    - spare no effort/expense
    - considers all possible join orders, including Cartesian products!

21

- Optimization requires processing time and memory
  - You can control resources applied to query optimization:
  - (similar to the -O flag in a C compiler)
  - Special register, for dynamic SQL
    - SET CURRENT QUERY OPTIMIZATION = 1
  - Bind option, for static SQL
    - BIND YOURAPP.BND QUERYOPT 1
  - Database configuration parameter, for default
    - UPDATE DB CFG FOR <DB> USING DFT_QUERYOPT <N>
  - Static & dynamic SQL may use different values

DB2_REDUCED_OPTIMIZATION - YES

- Optimization level == 5
  - Disables some optimization techniques that might consume significant prepare time and resources but do not usually produce a better access plan
    - Falls back to Greedy join enumeration in more cases
    - Limits the number of outer plans considered when planning joins
    - Avoids expensive analysis to determine the order property of the of each operator
    - Avoids repartitioned joins in a DB partitioned system
    - Limits the number of tables on a join inner
  - 'Automatic' optimization level, so more techniques considered for disabling
- Optimization level < 5
  - Avoids repartitioned joins in a DB partitioned system
  - Some of the above techniques aren't in effect anyway
  - Prepare time is usually less of a problem for these optimization levels
- Optimization level > 5
  - Not applicable
  - Typically for complex queries
  - Expect to tradeoff prepare time for a more optimal access plan

22

First, try setting the registry variable to YES. If the optimization class is 5 (the default) or lower, the optimizer disables some optimization techniques that might consume significant prepare time and resources but that do not usually produce a better query access plan. If the optimization class is exactly 5, the optimizer reduces or disables some additional techniques, which might further reduce optimization time and resource use, but also further increase the risk of a less than optimal query access plan. For optimization classes lower than 5, some of these techniques might not be in effect in any case. If they are, however, they remain in effect.

DB2_REDUCED_OPTIMIZATION – <number> | DISABLE

- <number>
  - Applies to dynamically prepared queries at optimization level 5
  - If the number of joins in **any** sub-select is > number, Greedy join enumeration is used for entire query
    - Based on sub-select as seen in the *optimized SQL* statement shown by explain
  - Plus, same effect as YES
  - This basically makes the query optimization level 2
- DISABLE
  - Disables the heuristic to fallback to Greedy at optimization level 5 when complex query AND (bufferpool < 2000 pages OR CPUSPEED > 3.5e-5)

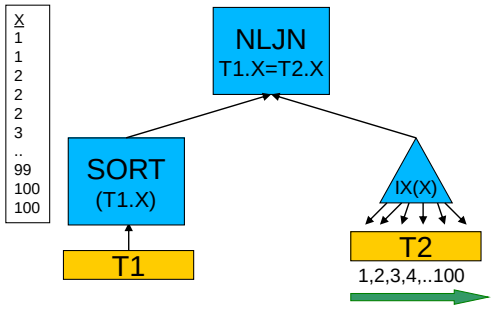
DB2_REDUCED_OPTIMIZATION – NO_SORT_NLJOIN

- Avoids sorting on outer stream join columns for *nested-loop join method*
- Order on join columns can sometimes help NLJOIN performance
  - When there is an index on the join column on the inner
  - Equality join predicates
  - Allows the inner table to be accessed in index order, which can allow better I/O prefetching
  - *Ordered NLJOIN*
  - This is sometimes a difficult costing decision
  - The additional SORT might have too much overhead for an OLTP system
- SORTs for other order requirements are still considered
  - e.g. ORDER BY, GROUP BY, DISTINCT
  - It is cheaper to do the SORT before expanding joins

24

Ordered Nested-loop Join

DB2_REDUCED_OPTIMIZATION=NO_SORT_NLJOIN
avoids this SORT



- Clustering: 98%
- T2 is accessed in a single, sequential scan
- Sequential prefetch detection doesn't need to be performed for each outer row
- Also works if data page FETCH is required

25

When it evaluates a nested loop join, the optimizer also decides whether to sort the outer table before performing the join. If it orders the outer table, based on the join columns, the number of read operations to access pages from disk for the inner table might be reduced, because they are more likely to be be in the buffer pool already. If the join uses a highly clustered index to access the inner table and if the outer table has been sorted, the number of index pages accessed might be minimized.
In addition, if the optimizer expects that the join will make a later sort more expensive, it might also choose to perform the sort before the join. A later sort might be required to support a GROUP BY, DISTINCT, ORDER BY or merge join.

DB2_REDUCED_OPTIMIZATION – NO_SORT_MGJOIN

- Avoids sorting on join columns for *merge join method*
- Order on join columns is necessary for MGJOIN
  - The order can be provided by SORT or by indexes
  - SORTs might have too much overhead for an OLTP system
  - Costing *usually* favours using indexes over SORT
  - But sometimes costing gets it wrong
  - If there are no indexes on join columns, MSJOIN is not possible with this option
- SORTs for other order requirements are still considered, if they include the join columns
  - e.g. ORDER BY, GROUP BY, DISTINCT
  - Need to SORT anyway if there are no indexes, so might as well try MGJOIN

DB2_REDUCED_OPTIMIZATION – INDEX (1|5)

- Enables heuristics for choosing indexes
  - Overrides normal costing decision
- Heuristic strategy:
  - Pick "obvious winners" and discard "obvious losers"
  - Minimize locking by avoiding reading rows that don't qualify predicates
  - Avoid making worse decisions than costing
    - Heuristics are relatively conservative
  - Applied consistently
    - e.g If index IX1 > IX2, and IX2 > IX3, then IX1 > IX3
- Undocumented option
  - Set under DB2_WORKLOAD=SAP, so maybe someday

27

DB2_REDUCED_OPTIMIZATION – INDEX (2|5)

- Rules are used for choosing among non-fully qualified unique IXSCANs and TBSCANs
- Fully qualified unique IXSCAN:
  - CREATE UNIQUE INDEX IX1 (A,B)
  - WHERE A=? AND B=?
  - Always beats a non-fully qualified unique IXSCAN or TBSCAN
  - Index-only plans favoured
- Rules consider:
  - Whether table is missing statistics or marked VOLATILE
  - Number of predicates
  - How predicates are applied
    - start/stop key preferred over sargable
  - Index-only access vs FETCH-IXSCAN
  - Does the index provide an 'interesting' order
    - Avoids SORTs

28

DB2_REDUCED_OPTIMIZATION – INDEX (3|5)

- The exact rules will not be provided in this presentation 
- But here are some examples
- Example 1:
  - IX1 (A,B,C) IX2 (A,B,D)
  - SELECT * FROM T WHERE A=? AND B=? AND C=?
  - IX1 wins because it applies more start/stop key predicates
  - TBSCAN not considered because IX1 applies all predicates

29

DB2_REDUCED_OPTIMIZATION – INDEX (4|5)

- Example 2:
  - IX1(A,B,C,D), IX2(A,C)
  - SELECT A,C,D FROM T WHERE A=? AND C=?
  - If T is missing statistics or marked VOLATILE
    - IX2 wins because it can apply more start/stop key predicates
    - TBSCAN not considered
  - ELSE
    - Tie. Even though IX2 applies more start/stop key predicates, IX1 avoids a FETCH
    - TBSCAN is not considered because both indexes apply all predicates
- Example 3:
  - IX1(A,B,C), IX2(D,E)
  - SELECT * FROM T WHERE A=? AND B=? AND C=? AND D=? AND E=?
  - IX1 wins because it can apply more start/stop key predicates
  - TBSCAN is considered because IX1 doesn't apply all predicates, unless T is missing statistics or marked VOLATILE

30

DB2_REDUCED_OPTIMIZATION – INDEX (5|5)

- Consider it for:
  - OLTP systems
  - Tables with many indexes

DB2_INLIST_TO_NLJN

- The optimizer can convert IN-list predicates into joins

Original:

```
SELECT * FROM T WHERE C IN (10,20,30,40)
```

Rewritten:

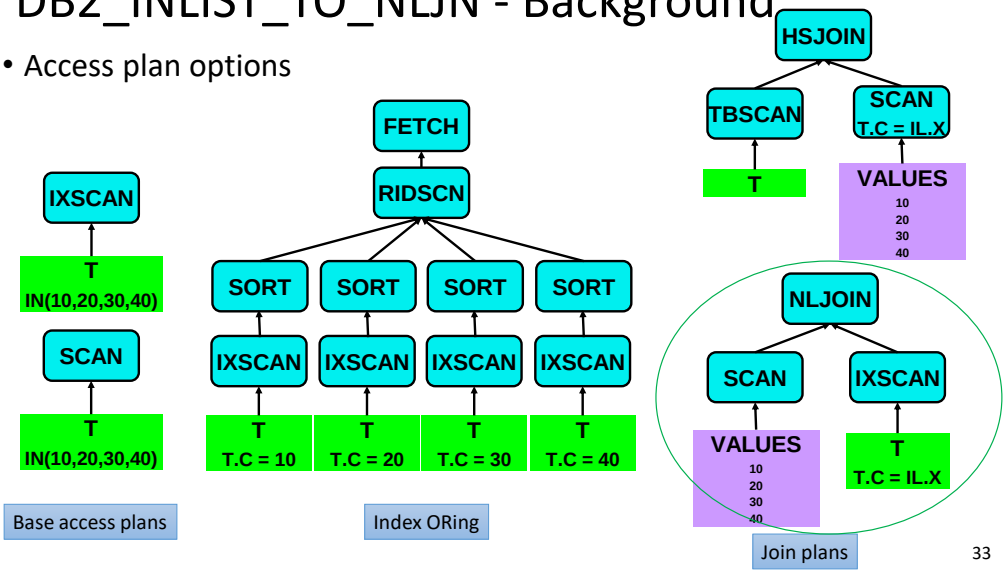
```
SELECT T.* FROM T,
  (SELECT DISTINCT X FROM (VALUES 10,20,30,40) AS
V(X)) AS IL(X) WHERE T.C = IL.X
```

- The optimizer will consider several access plan options
- The join version could be more efficient

32

DB2_INLIST_TO_NLJN - Background

- Access plan options



33

DB2_INLIST_TO_NLJN – [YES | ALL]

- Favours the NLJOIN plan if the predicate(s) can be applied as start/stop keys
- **YES**: Heuristic only applies if the IN-list uses **input variables**
  - The selectivity estimate could be inaccurate because the IN-list values aren't known at optimization time
- **ALL**: Heuristic applies if the IN-list contains input variables or literals
  - Undocumented option

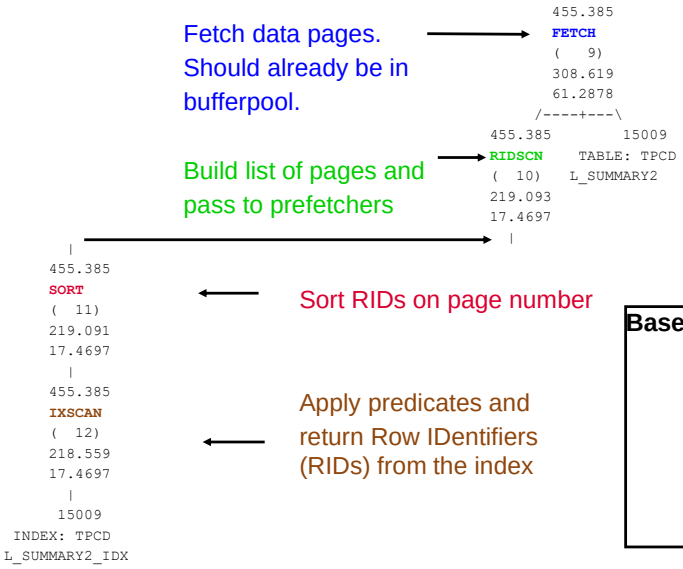
- Consider using the REOPT option instead

DB2_MINIMIZE_LISTPREFETCH

- "List prefetch" is a special type of index access method
- Used when the table's data pages need to be accessed but the pages are poorly clustered
- Approach:
  - Retrieve the qualifying row IDs (RIDs) from the index
  - Sort them by page number
  - Prefetch the data pages
- Optimizer has a detailed cost model for list prefetch
- But it requires accurate cardinality estimates and index statistics

35

List Prefetch Access Plan



Based on db2exfmt output

Cardinality (rows)
OPERATOR NAME
(Operator ID)
Cost (timerons)
I/O (pages)

List Prefetch is a method the optimizer uses to order the pages that must be fetch from a table, so that each page need only be touched once, and the table accessed in a sequential order, from start to finish. List Prefetch initiates simultaneous I/O to bring the pages of interest into the bufferpool, as quickly as possible. The increases I/O and processing overlap, and thus provides a considerable speedup, but at the cost of increased resource consumption (more parallel I/O).

DB2_MINIMIZE_LISTPREFETCH - YES

- Sometimes the optimizer can't cost list prefetch accurately
  - Common theme here – inaccurate selectivity estimates
- List prefetch might be the best standalone choice, but the RID SORT can produce too much overhead and consume too much memory on OLTP systems, due to high concurrency
- YES: Heuristic only applies if the predicates use input variables
- There is no ALL option 
- DB2_REDUCED_OPTIMIZATION=INDEX will also avoid using list prefetch, but without the input host variable requirement

37

Setting SQL Compiler Registry Variables (1|2)

- For db2set, changes take effect:
  - At next statement compilation for dynamic SQL
    - Use **FLUSH PACKAGE CACHE** in order to force immediate application
  - When package is rebound for static SQL
  - Use **ENV_GET_REG_VARIABLES** table function to retrieve registry variable values
    - Returns ‘in-memory’ and ‘on disk’ values
    - Can be different for static registry variables that are changed before the instance is restarted

38

Setting SQL Compiler Registry Variables (2|2)

- Can be specified at the global or statement level using *optimization profiles*
  - Allows control at the package or statement level
- Usage is identified in the explain output
  - ENVVAR argument of the RETURN plan operator:

```
ENVVAR   : (Environment Variable)
       DB2_EXTENDED_OPTIMIZATION=YES
       DB2_SELECTIVITY=YES[Global Optimization Guideline]
```

Set via db2set

- TIP: Use with a connect proc to limit SQL compiler registry variables based on query environment
  - E.g. user, various client attributes, etc.

39

Optimization profiles can have different registry variable values applied to a specific query statement or to many query statements used in an application.

Setting registry variables in an optimization profile can increase the flexibility you have in using different query statements for different applications. When you use the db2set command to set registry variables, the registry variable values are applied to the entire instance. In optimization profiles, the registry variable values apply only to the statements specified in the optimization profile. By setting registry variables in an optimization profile, you can tailor specific statements for applications without worrying about the registry variable settings of other query statements.
Only a subset of registry variables can be set in an optimization profile.
See here for the full list:
https://www.ibm.com/docs/en/db2/11.5?topic=profiles-sql-compiler-registry-variables-in-optimization-profile

Connect Procedure

- Use a connect procedure to put an optimization profile in effect
  - Stored procedure that is executed when DB connection is established
- Avoids issuing an explicit SET CURRENT OPTIMIZATION PROFILE within each connection
- Useful for setting SQL compiler registry variables
- Can include conditions to set different profiles based on user or client information

```
-- Create the connection procedure
CREATE PROCEDURE DBA.CONNECTPROC ( )
READS SQL DATA
LANGUAGE SQL
 if (session_user like 'APP1%' then
   set current optimization profile 'OPTPROF_APP1'
 elseif
   set current optimization profile 'OPTPROF_APP2'
 end if @

-- Register the connection procedure in the DB config
db2 update db cfg for <dbname> using connect_proc "DBA.CONNECTPROC";
```

40

The connect procedure provides you a way to allow applications in your environment to implicitly execute a specific procedure upon connection. This procedure can allow you to customize an application environment to a database from a central point of control. For example, in the connect procedure you can set special registers such as CURRENT_PATH to non-default values by invoking the SET CURRENT PATH statement. This new CURRENT_PATH value will now be the effective default CURRENT_PATH for all applications.

Any procedure created in the database that conforms to the naming and parameter restrictions can be used as the connect procedure for that database. The customization logic is provided by you in the form of a procedure created in the same database and is allowed to do any of the usual actions of a procedure such as issue SQL statements.
https://www.ibm.com/docs/en/db2/11.5?topic=databases-customizing-application-environment-using-connect-procedure

Summary

- Db2 supports a wide variety of methods for query execution
- The optimizer has a sophisticated cost model for query execution
- Sometimes the optimizer's estimates are inaccurate
- Sometimes it can take a long time for the optimizer to produce an access plan
- There are SQL compiler registry variables that might help specific situations or workloads