



## **Desperate Measures - How to Relieve the Db2 Optimizer of its Duties**

**John Hornibrook**  
*IBM*

Optimal query access plans are essential for good data server performance, and it is the query optimizer's job to choose the best one. However, occasionally no amount of statistics or tuning is enough to get the access plan you want. Or your application is effectively down, due to a poorly-performing query, and there isn't time to implement best practices. When these situations arise, Db2 optimization profiles and guidelines can be used to correct the access plan and get your application performing well again, quickly. Optimization profiles can specify various aspects of an access plan, can control automatic query rewriting or can control the optimizer's plan search space. Guidelines can be specified with or without modifying your application. This presentation will show you what options are available and how to use them effectively.

# Optimization profiles

- More commonly known as 'hints'
- Ability to specify access plan details
  - Index scan, join method, join order, etc.
- Ability to control statement optimization
  - Can control both query rewrite optimization and access plan optimization
- Can be put into effect without editing application code
  - Compose optimization profile, add to DB, rebind targeted packages
- **Should only be used after all other tuning options exhausted**

2

The DB2 data server supports an even more direct way to influence access plans using optimization profiles. Optimization profiles allow you to specify access plan details such as base access and join methods and join order. For example, you can specify that access to a particular table should use a particular index or you can specify that two tables should be joined using the hash join method. Optimization profiles also allow you to control query rewrite optimizations such transforming certain types of subqueries to joins. You can specify the base table access methods, join methods, and join order for the entire access plan, or just a subset of the access plan.

Optimization profiles are a powerful tool for controlling access plans; however, they should be used with caution. Optimization profiles prevent access plans from adjusting to changes in your data and your environment. While this does result in more stable access plans, it may be a bad approach when used for extended periods of time, because the performance improvements resulting from better access plans will never be realized. Optimization profiles are best used for exceptional situations when the tuning actions described previously in the presentation are unsuccessful in improving or stabilizing access plans.

# Optimization profiles: anatomy

- An **XML document** stored in a special system table
- Elements and attributes understood as explicit optimization guidelines/hints
- Composed and validated with an XML schema  
[sqllib/misc/DB2OptProfile.xsd](#)
- **Profile Header** (exactly one)
  - Meta data and processing directives
- **Global optimization guidelines** (at most one)
  - Applies to all statements for which profile is in effect
  - E.g. eligible MQTs guideline defining MQTs to be considered for routing
- **Statement-level optimization guidelines** (zero or more)
  - Applies to a specific statement for which profile is in effect
  - Specifies aspects of desired access plan

3

An optimization profile is specified as an XML document that you create and store in the SYSTOOLS.OPT\_PROFILE table.

An optimization profile contains optimization guidelines that specify the access plan details. An optimization profile can contain optimization guidelines for one or more SQL statements. The SQL statement text is stored in the optimization profile along with the optimization guidelines. When an optimization profile is in effect for your application, each SQL statement compiled by your application will be matched to the SQL statements specified in the optimization profile. When a matching SQL statement is found in the optimization profile, the SQL compiler will use the optimization guidelines for that SQL statement while optimizing it.

# Profiles vs. guidelines

- Optimization profile:
  - Refers to the entire XML document that can contain various types of optimization guidelines for one or more SQL statements
  - Identified by the `<OPTPROFILE>` element
  - Stored in SYSTOOLS.OPT\_PROFILE
- Optimization guidelines
  - Refers to a portion of the optimization profile that provides hints or guidelines for specific aspects of query optimization
  - Identified by the `<OPTGUIDELINES>` element
  - Can be specified as an `SQL statement comment`

# Sample optimization profile

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="11.5.0">
  <!--
    Global optimization guidelines section.
    Optional but at most one.
  -->
  <OPTGUIDELINES>
    <MQT NAME="DBA.AvgSales"/>
    <MQT NAME="DBA.SumSales"/>
  </OPTGUIDELINES>
  <!--
    Statement profile section.
    Zero or more.
  -->
  <STMTPROFILE ID="Guidelines for TPCD Q9">
    <STMTKEY SCHEMA="TPCD">
      <![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P.PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39
AND P.P_TYPE = 'BRASS' AND S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(P.S1.PS_SUPPLYCOST) FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND S1.S_NATION = S.S_NATION)]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <IXSCAN TABID="Q1" INDEX="I_SUPPKEY"/>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

Use CDATA to prevent > and < from  
being interpreted by the XML parser

# Optimization Guidelines (1 | 2)

- Access plan guidelines
  - Base access request
    - Method to access a table e.g. TBSCAN, IXSCAN
  - Join request
    - Method and sequence for performing a join e.g. HSJOIN, NLJOIN, MSJOIN
    - IXAND star joins
- Query rewrite guidelines
  - IN-list to join
  - Subquery to join
  - NOT EXISTS subquery to anti-join
  - NOT IN subquery to anti-join

# Optimization Guidelines (2 | 2)

- General optimization guidelines
  - REOPT (ONCE/ALWAYS/NONE)
    - Same as REOPT bind option
  - DEGREE
    - Intra-partition query parallelism degree
  - QRYOPT
    - Query optimization level
  - RTS
    - Real-time statistics, enable/disable, timeout time
  - MQT choices
    - Materialized query table options



# Access Plan Optimization Guidelines

- **Example:**

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpcd".PARTS, "Tpcd".SUPPLIERS, "Tpcd".PARTSUPP PS
WHERE
  P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P_SIZE = 39 AND P_TYPE = 'BRASS'
  PS.PS_SUPPLYCOST =
    (SELECT MIN(PS1.PS_SUPPLYCOST)
     FROM "Tpcd".PARTSUPP PS1, "Tpcd".SUPPLIERS S1
     WHERE "Tpcd".PARTS.P_PARTKEY = PS1.PS_PARTKEY AND
           S1.S_SUPPKEY = PS1.PS_SUPPKEY AND S1.S_NATION = S.S_NATION)
ORDER BY S.S_NAME
/* <OPTGUIDELINES><IXSCAN TABLE='S' INDEX='I_SUPPKEY' /></OPTGUIDELINES> */
```

- Choose an index access using index 'I\_SUPPKEY' for access to SUPPLIERS table in main sub-select
  - **Don't specify an index qualifier/schema! (It will not be recognized)**
- Table is referenced using correlation name 'S'
  - TABLE attribute must reference the 'exposed' name



# Access Plan Optimization Guidelines

- **Example:**

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
FROM "Tpced".PARTS, "Tpced".SUPPLIERS S, "Tpced".PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND
P_SIZE = 39 AND P_TYPE = 'BRASS'
PS.PS_SUPPLYCOST =
(SELECT MIN(PS1.PS_SUPPLYCOST)
 FROM "Tpced".PARTSUPP PS1, "Tpced".SUPPLIERS S1
 WHERE "Tpced".PARTS.P_PARTKEY = PS1.PS_PARTKEY AND
 S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
 S1.S_NATION = S.S_NATION)
ORDER BY S.S_NAME
/* <OPTGUIDELINES>
  <NLJOIN>
    <IXSCAN TABLE="Tpced".Parts' />
    <IXSCAN TABLE="PS" />
  </NLJOIN>
</OPTGUIDELINES> */
```

Index name not provided so optimizer  
chooses based on cost

- Join requests contains 2 elements – inner and outer
- Elements can be base accesses or other join requests

# Nesting join requests

- Example

<OPTGUIDELINES>

<MSJOIN>

<NLJOIN>

<IXSCAN TABLE='Tpcd'.Parts' />

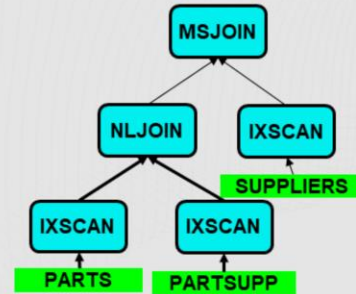
<IXSCAN TABLE="PS" />

</NLJOIN>

<IXSCAN TABLE='S' />

</MSJOIN>

</OPTGUIDELINES>



- Nested loop join is on outer of merge scan join
- Nested access request elements inside a join request must reference tables in the **same FROM clause** of the optimized statement
  - (more on this later)
- Query rewrite optimization guidelines, as well as general optimization guidelines, can affect the optimized statement.

# Forming table references

- 2 methods
  - Reference 'exposed' name in the original SQL statement
    - Use 'TABLE' attribute
    - Rules for specifying SQL identifiers apply to 'TABLE' attribute
  - Reference correlation name in the optimized SQL statement
    - Use 'TABID' attribute
    - 'Optimized' SQL is the semantically equivalent version of the statement after it has been optimized by query rewrite
    - Use the explain facility to get the optimized SQL statement
    - NOTE: There is no guarantee that correlation names in the optimized SQL statement are stable across new releases
- Table references must refer to a single table or they are ignored
  - i.e. no ambiguous references
- Unqualified table references are implicitly qualified by the current schema
- If both 'TABLE' and 'TABID' are specified, they must refer to the same table or they are ignored.
- Use 'TABID' to reference derived tables

# Referencing derived tables

- Example

```
SELECT S.S_NAME, PS.PS_AVAILQTY, P.P_NAME
FROM TPCD.SUPPLIER S INNER JOIN TPCD.PARTSUPP PS
ON S.S_SUPPKEY = PS.PS_SUPPKEY
LEFT OUTER JOIN TPCD.PART P
ON P.P_PARTKEY = PS.PS_PARTKEY
```

- This guideline is **invalid** because the inner join between SUPPLIER and PARTSUPP is in a separate derived table :

```
<OPTGUIDELINES>
  <HSJOIN>
    <IXSCAN TABLE="P"/>
  <HSJOIN>
    <IXSCAN TABLE="S" />
    <IXSCAN TABLE="PS"/>
  </HSJOIN>
</HSJOIN>
</OPTGUIDELINES>
```

# Referencing derived tables

- Option 1: Reference derived table using TABID and optimized SQL correlation name
  - Doesn't require modifying the SQL text
  - But optimized SQL correlation names aren't stable across releases

<OPTGUIDELINES>

<HSJOIN>

<IXSCAN TABLE="P"/>

<ACCESS TABID="Q3" />

</HSJOIN>

<HSJOIN>

<IXSCAN TABLE="S" />

<IXSCAN TABLE="PS"/>

</HSJOIN>

</OPTGUIDELINES>

```
SELECT Q3.S_NAME AS "S_NAME", Q3.PS_AVAILQTY AS  
"PS_AVAILQTY", Q4.P_NAME AS "P_NAME"
```

```
FROM
```

```
(SELECT Q2.PS_AVAILQTY, Q2.PS_PARTKEY, Q1.S_NAME  
FROM
```

```
TPCD.SUPPLIER AS Q1,
```

```
TPCD.PARTSUPP AS Q2
```

```
WHERE
```

```
(Q1.S_SUPPKEY = Q2.PS_SUPPKEY)
```

```
) AS Q3
```

```
LEFT OUTER JOIN TPCD.PART AS Q4
```

```
ON (Q4.P_PARTKEY = Q3.PS_PARTKEY)
```

**Tip: Get the optimized SQL from the explain facility**

# Referencing derived tables

- Option 2: Rewrite the SQL to use an inline view and reference the view in the guideline:

```
WITH VX AS
(SELECT S.S_NAME, PS.PS_AVAILQTY, PS.PS_PARTKEY
 FROM TPCD.SUPPLIER S INNER JOIN TPCD.PARTSUPP PS ON S.S_SUPPKEY = PS.PS_SUPPKEY)
SELECT VX.S_NAME, VX.PS_AVAILQTY, P.P_NAME
FROM VX LEFT OUTER JOIN TPCD.PART P ON P.P_PARTKEY = VX.PS_PARTKEY
/* <OPTGUIDELINES>
<HSJOIN>
  <IXSCAN TABLE="P"/>
  <ACCESS TABLE="VX" />
</HSJOIN>
<HSJOIN>
  <IXSCAN TABLE="S" />
  <IXSCAN TABLE="PS"/>
</HSJOIN>
</OPTGUIDELINES> */
```

Must also unnest the join guidelines

Caveat – works for this example, but it  
might not in more complex scenarios

# Table references in views

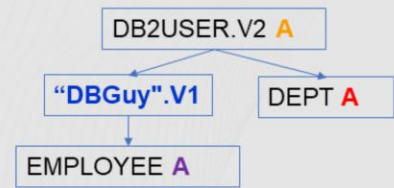
- **Example**

```
CREATE VIEW "DBGuy".V1 as  
(SELECT * FROM EMPLOYEE A WHERE SALARY > 50000);
```

```
CREATE VIEW DB2USER.V2 AS (SELECT * FROM "DBGuy".V1, DEPT A  
WHERE A.MGR_ID IN ('52', '53','54') AND "DBGuy".V1.DEPTNO = A.DEPTNO);
```

```
SELECT * FROM DB2USER.V2 A WHERE A.HIRE_DATE > '01/01/2004'  
/* <OPTGUIDELINES><IXSCAN TABLE=A/"DBGuy".V1/A"/></OPTGUIDELINES> */
```

- Extended syntax allows unambiguous table references in views
  - `TABLE='A'` is ambiguous and would return an error
- Extended name consists of exposed names in the path, from the statement reference to the nested reference, separated by slashes
- Same rules for exposed names apply to extended syntax





# Table references in views

- Extended syntax is not necessary if all exposed names for table references are unique

- **Example**

```
CREATE VIEW "DBGuy".V1 as (SELECT * FROM EMPLOYEE E WHERE  
    SALARY > 50,000) ;
```

```
CREATE VIEW DB2USER.V2 AS (SELECT * FROM "DBGuy".V1 WHERE  
    DEPTNO IN ('52', '53','54') ;
```

```
SELECT * FROM DB2USER.V2 A WHERE V2.HIRE_DATE > '01/01/2004'  
/* <OPTGUIDELINES><IXSCAN TABLE='E'></OPTGUIDELINES> */ ;
```

# Ambiguous table references

- **Example**

```
CREATE VIEW V1 AS
```

```
(SELECT * FROM EMPLOYEE WHERE SALARY >  
  (SELECT AVG(SALARY) FROM EMPLOYEE));
```

```
SELECT * FROM V1 WHERE DEPTNO IN ('M62', 'M63')
```

```
/* <OPTGUIDELINES><IXSCAN TABLE='V1/EMPLOYEE'></OPTGUIDELINES>* /;
```

- Which EMPLOYEE reference?
- The IXSCAN request is ignored
- Uniquely identify EMPLOYEE by adding correlation names in the view
- Use TABID
  - Correlation names in the optimized SQL are always unique

# Conflicting optimization guidelines

- Example

**<OPTGUIDELINES>**

**<IXSCAN TABLE='Tpcd'.PARTS' INDEX='I\_PTYPE'/>**

**<IXSCAN TABLE='Tpcd'.PARTS' INDEX='I\_SIZE'/>**

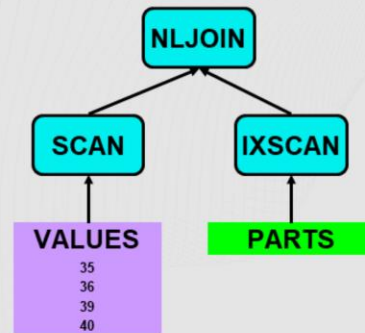
**</OPTGUIDELINES>**

- Multiple optimization guidelines can't reference the same table
- The first reference is applied and the others are ignored
- If I\_PTYPE doesn't exist but I\_SIZE does, the guideline is still ignored !

# Query Rewrite Guidelines

- **Example:**

```
SELECT S.S_NAME ...  
FROM  
  "Tpcd".PARTS P, "Tpcd".SUPPLIERS S, "Tpcd".PARTSUPP PS  
WHERE  
  P_PARTKEY = PS.PS_PARTKEY AND  
  S.S_SUPPKEY = PS.PS_SUPPKEY AND  
  P_SIZE IN (35, 36, 39, 40) AND  
  S.S_NATION IN ('INDIA', 'SPAIN')  
ORDER BY S.S_NAME  
/* <OPTGUIDELINES><INLIST2JOIN TABLE='P' /></OPTGUIDELINES> */
```



- INLIST2JOIN specifies that list of constants in IN list predicate should be transformed to an **in-memory table** (similar to a VALUES clause)
- In-memory table can then be joined to "Tpcd".PARTS using an indexed NLJN
  - Or a HSJN too. The join method can also be specified!
- Target IN-list identified by specifying table to which predicate is applied
- If there are multiple IN-lists for the same table, guideline can be further qualified with COLUMN attribute

# General Optimization Guidelines

- **Example**

```
SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE  
FROM "Tpcd".SUPPLIERS S  
WHERE S.S_NATION IN (?, ?) AND S.S_SUPPKEY = ?  
ORDER BY S.S_NAME
```

```
/* <OPTGUIDELINES> <REOPT VALUE='ONCE' /> </OPTGUIDELINES> */
```

- 'ONCE' indicates that optimization should be deferred until the first set of variable values is provided.
- This allows the optimizer to compare the input values to the statistics to get a better selectivity estimate and a better query execution plan

20

<https://www.ibm.com/docs/en/db2/11.5?topic=wtqop-using-reopt-bind-option-input-variables-in-complex-queries>

## REOPT

Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, global variables, and special registers. Valid values are:

### NONE

The access path for a given SQL statement containing host variables, parameter markers, global variables, or special registers will not be optimized using real values for these variables. The default estimates for these variables will be used instead, and this plan is cached and used subsequently. This is the default behavior.

### ONCE

The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, global variables, or special registers when the query is first executed. This plan is cached and used subsequently.

### ALWAYS

The access path for a given SQL statement will always be compiled and

reoptimized using the values of the host variables, parameter markers, global variables, or special registers known at each execution time.

# Putting an optimization profile into effect

- Create the OPT\_PROFILE table in the SYSTOOLS schema:

```
CALL SYSPROC.SYSINSTALLOBJECTS('OPT_PROFILES', 'C',  
    CAST (NULL AS VARCHAR(128)), CAST (NULL AS VARCHAR(128)))
```

- Alternatively, can issue this DDL directly:

```
CREATE TABLE SYSTOOLS.OPT_PROFILE (  
    SCHEMA VARCHAR(128) NOT NULL,  
    NAME VARCHAR(128) NOT NULL,  
    PROFILE BLOB (2M) NOT NULL,  
    PRIMARY KEY ( SCHEMA, NAME ));
```

- Compose document, validate, insert into table with qualified name

*Inserts inventory\_db.xml from current directory into the SYSTOOLS.OPT\_PROFILE table with qualified name "DBA"."INVENTDB"*

File profiledata:

```
"DBA","INVENTDB","inventory_db.xml"
```

```
IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE  
INSERT INTO SYSTOOLS.OPT_PROFILE;
```



# Putting an optimization profile into effect

- Alternative way to insert optimization profile:

```
insert into systools.opt_profile values(  
current schema, 'OPTPROF1', blob('<?xml version="1.0" encoding="UTF-8"?>  
<OPTPROFILE>  
  <STMTPROFILE ID="Test 1">  
    <STMTKEY>  
      <![CDATA[select * from ta where a1 > 10 and a1 < 20]]>  
    </STMTKEY>  
    <OPTGUIDELINES><IXSCAN TABLE="TA" INDEX="TA_IX1"/></OPTGUIDELINES>  
  </STMTPROFILE>  
</OPTPROFILE>') );
```

- Cast the input string to BLOB

# Putting an optimization profile into effect

- At the package level using the **OPTPROFILE** bind option
- For existing packages, use ALTER PACKAGE:

```
ALTER PACKAGE DB2USER.EMPADMIN OPTIMIZATION PROFILE DB2USER.JOINHINT
```

```
/* Bind optimization profile "DBA"."INVENTDB" to the package "inventapp" */  
DB2 PREP INVENTAPP.SQC BINDFILE OPTPROFILE DBA.INVENTDB  
DB2 BIND INVENTAPP.BND
```

- At the dynamic statement level: using **CURRENT OPTIMIZATION PROFILE** special register

```
/* Optimize statements using profile 'DBA.INVENTDB' */  
SET CURRENT OPTIMIZATION PROFILE = 'DBA.INVENTDB' ;
```

```
/* Optimize statements using profile 'JON.SALES' */  
SET CURRENT SCHEMA = 'JON' ;  
SET CURRENT OPTIMIZATION PROFILE = 'SALES' ;
```

# Putting an optimization profile into effect

```
/* Clear the special register and use the setting of the OPTPROFILE bind option */  
SET CURRENT OPTIMIZATION PROFILE = NULL;
```

```
/* Don't use optimization profile at all */  
SET CURRENT OPTIMIZATION PROFILE = "";
```

- At the dynamic statement level: using `db2_optprofile` CLI option

*-- After each successful connect to the SANFRAN database, the CLI client would issue the command:*

```
SET CURRENT OPTIMIZATION PROFILE=JON.SALES.  
  
[SANFRAN]  
DB2_OPTPROFILE JON.SALES
```

# Putting an optimization profile into effect

- 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 general optimization guidelines
- 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";
```

25

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>

# Updating an existing optimization profile

- Re-insert an updated XML document in SYSTOOLS.OPT\_PROFILE
- Optimization profiles are parsed and cached to reduce overhead when used for different statements
- Must flush the cache in order to pick up updated version:
  - Flush the entire cache:  
`FLUSH OPTIMIZATION PROFILE CACHE`
  - Flush a cache entry for a specific optimization profile  
`FLUSH OPTIMIZATION PROFILE CACHE SALES`

# Handling SQL in procedures

- Use **SET\_ROUTINE\_OPTS** to specify the optimization profile

```
CALL SET_ROUTINE_OPTS('OPTPROFILE DBA.INVENTDB ' ) %
```

```
CREATE PROCEDURE MY_PROC  
BEGIN  
    DECLARE CUR1 CURSOR FOR SELECT ...  
END %
```

- SQL might be modified during CREATE PROCEDURE processing
- Use explain facility or query system catalogs to get the modified SQL statements to include in optimization profile STMTKEY element for profile statement matching
  - See speaker notes for an example

27

STMTNO should be the line number in the source code of the CREATE PROCEDURE, relative to the beginning of the procedure statement (line number 1)

```
SELECT STMTNO, SEQNO, SECTNO, TEXT  
FROM SYSCAT.STATEMENTS AS S,  
     SYSCAT.ROUTINEDEP AS D,  
     SYSCAT.ROUTINES  AS R  
WHERE PKGSCHEMA = BSCHEMA  
AND PKGNAME = BNAME;  
AND BTYPE = 'K'  
AND R.SPECIFICNAME = D.SPECIFICNAME  
AND R.ROUTINESCHAME = D.ROUTINESCHEMA  
AND ROUTINENAME = ?  
AND ROUTINESCHEMA = ?  
AND PARM_COUNT = ?  
ORDER BY STMTNO
```

# Embedded optimization guidelines

- Optimization guidelines can be specified along with the SQL statement in an SQL comment
- Only one set of guidelines can appear in a `/* */` comment after the entire SQL statement
- Supported for static and dynamic SQL and SQL/PL procedures
- Enabled by default in DB2 11.1
  - Must set registry variable DB2\_OPTPROFILES=ON in prior releases

```
SELECT S.S_NAME FROM TPCD.SUPPLIER S WHERE S_NAME = 'XYZ CORP'  
/* <OPTGUIDELINES><IXSCAN TABLE='S' INDEX='S_IX1'/></OPTGUIDELINES> */
```

28

<https://www.ibm.com/docs/en/db2/11.5?topic=guidelines-embedded-optimization>  
General rules when using embedded optimization guidelines:

Embedded optimization guidelines can only be applied to Data Manipulation Language (DML) statements: the SELECT, INSERT, UPDATE, DELETE, and MERGE commands. The optimizer will ignore such comments on other types of statements. No error or warning will be provided.

The embedded optimization guideline must be provided after the SQL portion of the statement. They cannot appear inside subselects. However, other types of comments can be provided at the end of the statement before or after the optimization guideline.

The optimizer will look for one embedded optimization guideline comment for every DML statement. If there are multiple embedded optimization guideline comments, all of them are ignored and a warning is produced.

The optimization guideline must be written in well-formed XML. It cannot include extraneous text.



## Embedded optimization guidelines – precedence order (1 | 2)

- Embedded overrides identical optimization guidelines specified in the global section of an optimization profile e.g.

```
SELECT COUNT(*) FROM TAB1  
/* <OPTGUIDELINES> <REOPT VALUE='ONCE' /> </OPTGUIDELINES> */;
```

Takes precedence over this active optimization profile:

```
<?xml version="1.0" encoding="UTF-8"?>  
<OPTPROFILE VERSION="11.5.0">  
  <OPTGUIDELINES> <REOPT VALUE='ALWAYS' /> </OPTGUIDELINES>  
</OPTPROFILE>
```

29

Embedded optimization guidelines override identical optimization guidelines specified in the global section of an optimization profile.

## Embedded optimization guidelines – precedence order (2 | 2)

- Statement profiles override embedded guidelines e.g.

```
SELECT COUNT(*) FROM TAB1  
/* <OPTGUIDELINES> <REOPT VALUE='ONCE' /> </OPTGUIDELINES> */;
```

- Is ignored with this active optimization profile:

```
<?xml version="1.0" encoding="UTF-8"?>  
<OPTPROFILE VERSION="11.5.0">  
  <STMTPROFILE ID="STMTPROF1">  
    <STMTKEY> <![CDATA[SELECT COUNT(*) FROM TAB1]]> </STMTKEY>  
    <OPTGUIDELINES> <IXSCAN TABLE="TAB1"/> </OPTGUIDELINES>  
  </STMTPROFILE>  
</OPTPROFILE>
```

30

Optimization guidelines provided by way of a statement profile section of an optimization profile take precedence over embedded optimization guidelines. That is, if the CURRENT OPTIMIZATION PROFILE register contains the name of an optimization profile, and the specified optimization profile contains a matching statement profile for a statement with embedded optimization guidelines, then the embedded optimization guidelines are ignored by the optimizer.

# Inexact SQL Statement Matching (1 | 4)

- Default: SQL statement text must be an exact match, other than white space
- It is difficult to create optimization profiles for complex query workloads
  - Queries might have same 'shape' but only differ by literals in predicates
- Match predicates on same columns with same relational operators, but different literal values:
  - $\text{NAME} = \text{'Joe'} \leftrightarrow \text{NAME} = \text{'Bob'}$
  - $\text{IN}(1,2,3) \leftrightarrow \text{IN}(4,5,6)$
- Match IN-list predicates with different numbers of items:
  - $\text{IN}(1,2) \leftrightarrow \text{IN}(1,2,3)$
  - $\text{IN}(?,?) \leftrightarrow \text{IN}(?,?,?)$
- Match predicates with references to different host variables:
  - $A = :hv1 \leftrightarrow A = :hv2$

## Inexact SQL Statement Matching (2 | 4)

- Inexact matching applies to all literals in the statement
- Different special registers will not match
- Statements that won't match:

C1 BETWEEN 5 AND :HV

5 BETWEEN C1 AND C2

C1 IN (SELECT C1 FROM T1)

C1 IN (1,2,3)

C1 IN (C1, 1, 2)

C1 IN (C2, 1, 2)

A = 5

A = 5 + :HV

WITH RR

WITH RS

C2 < CURRENT TIME

C2 < '11:12:40'

C3 > CURRENT TIMESTAMP

C3 > '07/29/2010'

## Inexact SQL Statement Matching (3 | 4)

- Applies to XQuery too
  - Except for literals passed as function parameters representing SQL statement text or column names
  - XMLQUERY, XMLEXISTS and XMLTABLE are always exactly matched
- Inexact matching is specified using **STMTMATCH** attribute
- Can be specified at the global or statement level
  - Global: applies to all statements executed when profile is active
- Statement level specification takes precedence over global level

# Inexact SQL Statement Matching (4|4)

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE>
  <!--Global section -->
  <STMTMATCH EXACT='FALSE' />
  <!-- Statement level profile -->
  <STMTPROFILE ID='S1'>
    <STMTMATCH EXACT='TRUE' />
    <STMTKEY>
      <![CDATA[select t1.c1, count(*) from t1,t2 where t1.c1 =
t2.c1 and t1.c1 > 0]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <NLJOIN>
        <TBSCAN TABLE='T1' />
        <TBSCAN TABLE='T2' />
      </NLJOIN>
    </OPTGUIDELINES>
  </STMTPROFILE>
  <STMTPROFILE ID='S2'>
    <STMTKEY>
      <![CDATA[select * from t1 where c1 in (10,20,30)]]>
    </STMTKEY>
    (etc)
  </STMTPROFILE>
</OPTPROFILE>
```

- Statement **S1** will use exact matching
- Statement **S2** will use inexact matching

# Optimization Profile SQL Statement Matching

- Other flexible matching done by default:
  - Match any SQL keywords regardless of case e.g.  
`SELECT`  $\leftrightarrow$  `SeLeCt`
  - Match delimited and undelimited identifiers e.g.  
`"T1"`  $\leftrightarrow$  `T1`



# Specifying SQL Compiler Registry Variables

- SQL compiler registry variables only affect SQL compilation and are dynamic
  - i.e. no instance restart is necessary (db2set -im )

```
<OPTGUIDELINES>
  <REGISTRY>
    <OPTION NAME='DB2_ANTIJOIN' VALUE='EXTEND' />
    <OPTION NAME='DB2_REDUCED_OPTIMIZATION' VALUE='YES' />
  </REGISTRY>
</OPTGUIDELINES>
```

- Can be specified at the global or statement level
- Allows control at the package or statement level
- **TIP:** Use with a connect proc to limit SQL compiler registry variables based on query environment
  - E.g. user, various client attributes, etc.
- Usage is identified in the explain output
  - ENVVAR argument of the RETURN plan operator:

```
ENVVAR : (Environment Variable)
        DB2_ANTIJOIN=NO[Global Optimization Guideline]
```

## Optimization guideline construction – access requests

- Access requests – specify desired method for satisfying table reference
- Correspond to Db2 data access methods
  - TBSCAN
  - IXSCAN
  - LPREFETCH (list prefetch)
  - IXAND (index ANDing)
  - IXOR (index Oring)
  - ACCESS (any access type)

## Optimization profile construction – join requests

- Join requests – specify desired method and order for joining tables
- Contain access or other join requests
- Correspond to Db2 join methods
  - NLJOIN
  - HSJOIN
  - MSJOIN
  - JOIN (any join type)

# Optimization profile construction – rewrite requests

- Specify query rewrite transformations
- Correspond to Db2 query rewrite transformation rules
  - INLIST2JOIN (IN-list predicate to join)
  - SUBQ2JOIN (Subquery to join)
  - NOTEX2AJ (NOT EXISTS subquery to anti-join)
  - NOTIN2AJ (NOT IN subquery to anti-join)

# Problem determination

- Invalid optimization guidelines are ignored
- SQL code +437 reason code 13 is returned
- Doesn't prevent the SQL statement from running
- Use the explain facility to understand why guideline wasn't applied
- 2 Explain tables for diagnostics
  - [EXPLAIN\\_DIAGNOSTICS](#)
  - [EXPLAIN\\_DIAGNOSTICS\\_DATA](#)
- db2exfmt will provide the formatted messages
- A table function is also available
  - [EXPLAIN\\_GET\\_MSGS](#)
- Use explain facility to verify if optimization profiles have been applied:
  - Check arguments on RETURN operator:
    - [OPT\\_PROF](#) – name of profile in SYSTOOLS.OPT\_PROFILE
    - [STMTPROF](#) – name specified in ID attribute of STMTPROFILE element

# Problem determination

- Example diagnostic messages:

EXP0004W The optimization profile or embedded optimization guideline is either not well-formed or is invalid. Line number "554", character number "20".

EXP0012W Invalid access request. The index "index2" could not be found. Line number "573", character number "20".

EXP0009W Invalid access request. The table reference identified by the TABLE attribute could not be found. Line number "573", character number "20"

- Line number

- Relative to position within the XML document

- Character number

- Relative to beginning of line specified by Line number

# Summary

- Help the optimizer do its job for all queries by providing it good information:
  - Statistics
  - System configuration information
  - Input variable values
  - Schema information via DB constraints
- Tune specific queries, where the above approaches fail
- Use more direct approaches like optimization profiles as a last resort
- Optimization profiles provide a high degree of control over query transformations and the access plan