Explain Essentials for Query Optimization

John Hornibrook
IBM Canada

Platform: Db2/LUW

One of the most important factors in query performance is the access plan chosen by the Db2 query optimizer. Understanding the query access plan is essential to understanding query performance and the explain facility is the way to get the access plan. But SQL statements can be issued in different ways (dynamic, static, procedures, application programs) and access plans can exist in a variety of contexts (system catalogs, dynamic statement cache, monitoring tables), so getting the correct access plan for the correct statement can sometimes be a challenge. This presentation will cover the most important techniques for collecting and using explain information so that you can quickly and effectively understand your access plans in order to achieve and maintain the best possible query performance.

Objectives

- Understand where access plans can exist and how to explain them
- Learn how to get explain information about query execution
- Learn how to ensure that the explain contains all the important information
- Discuss some techniques for monitoring access plan changes over time
- Learn how to interpret and understand explain information

The explain facility – what is it?

- Internal phase of the optimizer that captures critical information used in selecting the query access plan
- Access plan information is written to a set of tables
- External tools to format explain table contents:
  - Db2 Data Management Console Visual Explain
    - GUI to render and navigate query access plans
  - db2exfmt
    - Text-based output from the explain tables
    - Command-line interface

They show the same information

3

The explain facility is used to display the query access plan chosen by the query optimizer to run an SQL statement. It contains extensive details about the relational operations used to run the SQL statement such as the plan operators, their arguments, order of execution, and costs. Since the query access plan is one of the most critical factors in query performance, it is important to be able to understand the explain facility output in order to diagnose query performance problems.

Explain information is typically used to:

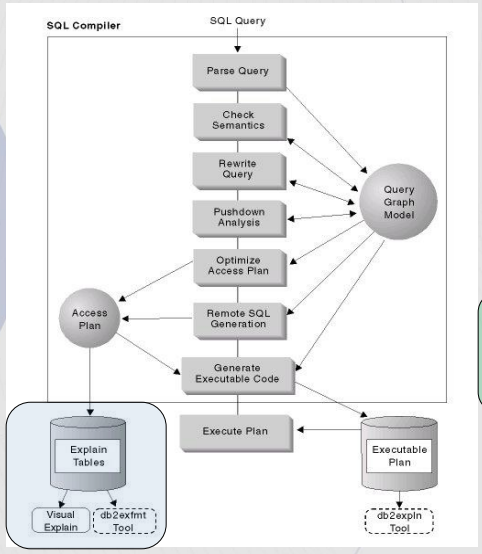
understand why application performance has changed

evaluate performance tuning efforts

The explain facility – why should you care?

- Access plan changes could degrade performance
- Explain helps to understand optimizer decisions
- Maintain a history of access plans for important SQL, during key transition periods
  - Major version upgrades
  - Significant DB or DB manager configuration changes
  - New index additions
  - Large data updates/additions
  - Catalog statistics changes
- Problem determination is easier, and often faster with a reference access plan to compare

Phases of SQL Compilation



Explain facility

- Sometimes references to 'optimization' really mean SQL compilation
- There is a lot more involved to SQL compilation

Parsing
- Catch syntax errors
- Generate internal representation of query

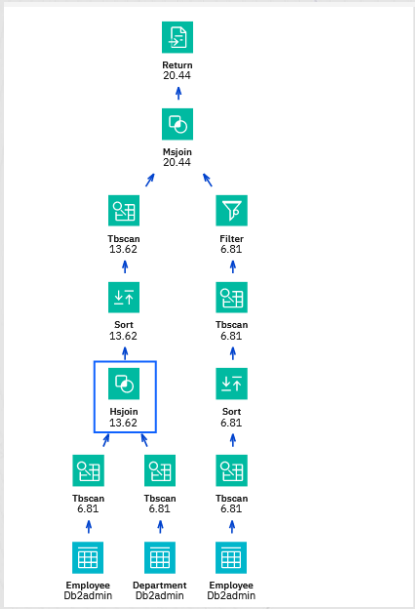
Semantic checking
- Determine if query makes sense
- Incorporate view definitions
- Add logic for constraint checking and triggers

Query optimization
- **Modify query to improve performance (Query Rewrite)**
- **Choose the most efficient "access plan"**

Pushdown Analysis
- Federation "optimization"

Threaded code generation
- Generate efficient "executable" code
- "Access section"

Db2 Data Management Console Visual Explain



Description

Properties

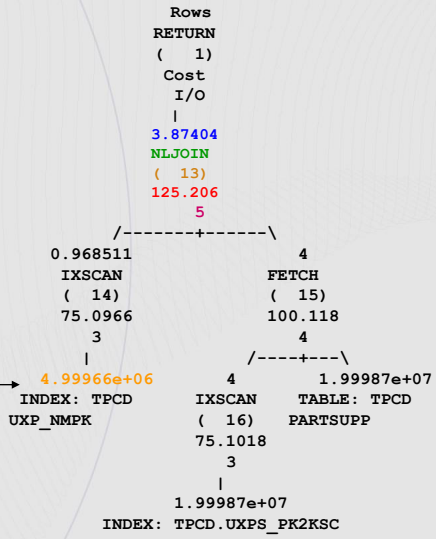
Name	Value
Operator identifier	5
Predicate text	(Q1.MGRNO = Q2.EMPNO)
Operator type	Hash join
Early out flag	RIGHT
Hash join bit filter used	FALSE
Temporary table page size	32768
Hash code size	24 BIT
HASHTBSZ	9
TUPBLKSZ	4000
Cost Information	
Estimated output cardinality	14.00
Cumulative total cost	13.62
Cumulative CPU cost	214,641.66
Cumulative I/O cost	2.00
Cumulative first row total	13.62

6

db2exfmt

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

Base table cardinality



Creating the Explain Tables

- Visual Explain tools create the explain tables automatically
- Best manual option: use SYSINSTALLOBJECTS stored procedure:

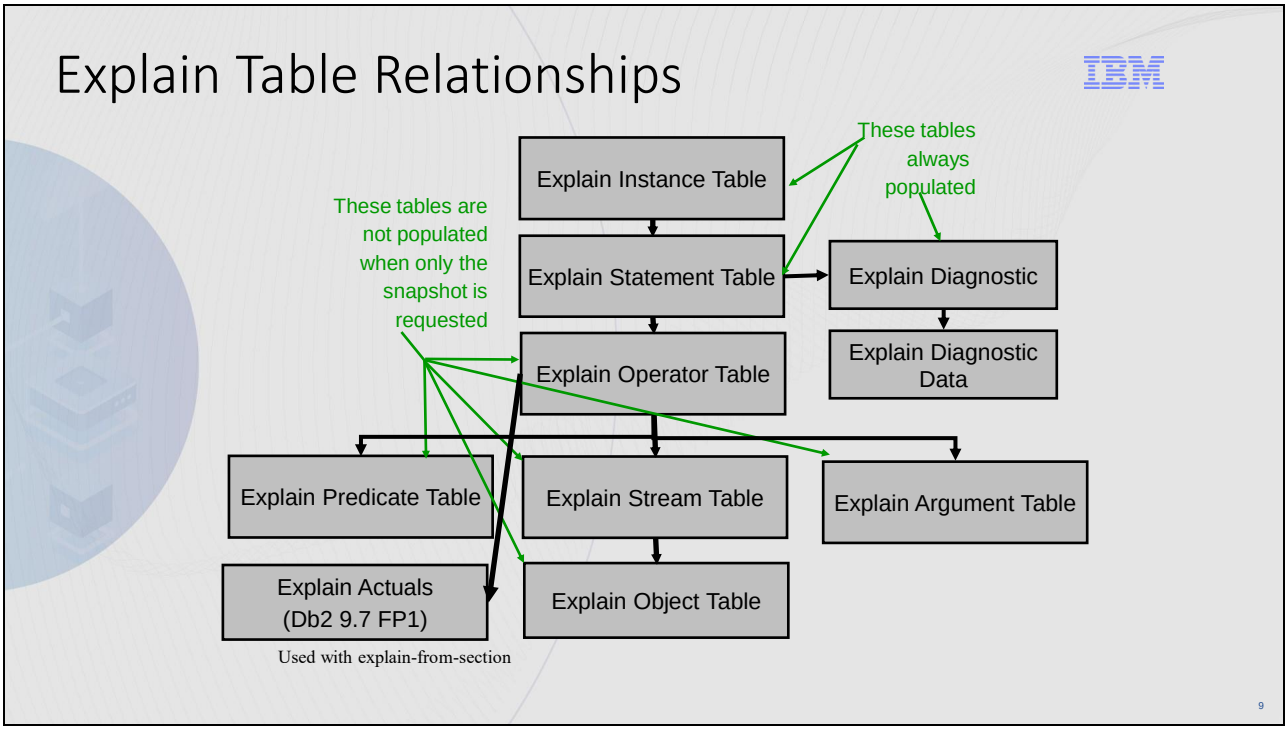
```
CALL SYSPROC.SYSINSTALLOBJECTS
('EXPLAIN', 'C', CAST (NULL AS VARCHAR(128)), 'DB2USER' )
```

- Alternative: create the explain tables manually using sqllib/misc/EXPLAIN.DDL:
  - `db2 -tvf ~/sqllib/misc/EXPLAIN.DDL`
- There are 10 explain tables
- Db2 inserts details of selected plan into the explain tables
- Details in Db2 documentation:

https://www.ibm.com/docs/en/db2/11.5?topic=sql-explain-tables

8

Calling the SYSPROC.SYSINSTALLOBJECTS procedure is preferred over using the EXPLAIN.DDL file since it can automatically adapt to different database configurations. For example, if BLOCKNONLOGGED parameter is set to yes, then some statements in EXPLAIN.DDL fail because NOT LOGGED clause is used for LOB columns. However, if BLOCKNONLOGGED parameter is set to yes then the SYSPROC.SYSINSTALLOBJECTS procedure automatically avoids the use of NOT LOGGED clause.



There are 10 explain tables, and 6 index advisor tables. This talk will not touch upon the index advisor tables.
The **Explain Instance** table contains information about a grouping of explain statements. This grouping is usually for static SQL statements that are part of the same source file.
The **Explain Statement** table contains information about a specific statement. There are 2 entries in this table for each SQL statement explain: the original statement, and the planned statement. Associated with the planned statement, is the rest of the Query Access Plan.
The **Explain Operator** table contains information about each of the operations in the Query Access Plan.
The **Explain Stream** table describes how each of the operators are linked together, and link to **Explain Objects**.
The **Explain Objects** tables describes all of the objects (indexes, tables, table functions) used by the Query Access Plan
The **Explain Predicate** tables describes all of the predicates applied by a particular **Explain Operator**.
The **Explain Argument** table describes additional arguments for the operator.
The **Explain Diagnostic** table contains diagnostic information about the statement such as tables or indexes that are missing statistics or why an MQT or statistical view wasn't used.
The **Explain Diagnostic Data** table contains multiple rows for each diagnostic message. These rows contain message tokens associated with the message.
The **Explain Actuals** table contains runtime actuals information. Currently, it contains the actual number of rows processed by each operator in the EXPLAIN_OPERATOR table.

How is explain information collected?

- There are many different ways to collect explain information!
- This presentation will cover most of them
- There are many ways because SQL can be issued in different contexts:
  - Static, dynamic
  - Stored procedures, triggers and functions
- The most common methods:
  - Use the EXPLAIN statement:
    - EXPLAIN PLAN FOR SELECT … FROM T1, T2 WHERE…
  - Use the CURRENT EXPLAIN MODE special register
    - SET CURRENT EXPLAIN MODE EXPLAIN
    - SELECT … FROM T1, T2 WHERE…
    - SET CURRENT EXPLAIN MODE NO

EXPLAIN statement vs EXPLAIN MODE special register?

- The EXPLAIN statement is useful if you want to uniquely identify each instance of an explained statement:

```
EXPLAIN PLAN SET QUERYNO = 13 SET QUERYTAG = 'TEST13'
FOR SELECT … FROM T1, T2 WHERE…
```

  - Values are stored in EXPLAIN_STATEMENT QUERYNO and QUERYTAG columns
- The EXPLAIN MODE special register is useful when:
  - You want to explain a set of SQL statements without pre-pending EXPLAIN PLAN:

```
SET CURRENT EXPLAIN MODE EXPLAIN
<stmt 1>
<stmt 2>
SET CURRENT EXPLAIN MODE NO
```

  - Want to explain AND execute the statements

```
SET CURRENT EXPLAIN MODE YES
```

Avoid db2expln and dynexpln

- These are tools that interpret the compiled version of the statement (access section)
- Contains less optimizer information about the query access plan
- Insufficient details about **why** operations were chosen
  - No predicate selectivity estimates, no query transformation information
- dynexpln was discontinued in Db2 10.1

12

https://www.ibm.com/docs/en/db2/11.5?topic=facility-tools-collecting-analyzing-explain-information

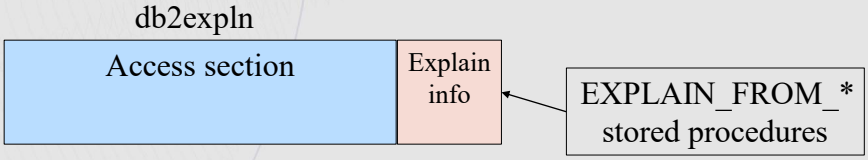
db2expln shows the actual implementation of the chosen access plan but it does not include as much optimizer information as db2exfmt. By examining the generated access plan, the db2expln tool provides a relatively compact, verbal overview of the operations that will occur at run time. However, the EXPLAIN_FROM_SECTION and EXPLAIN_FROM_DATA stored procedures are the recommended way to collect the explains while a query is running. EXPLAIN_FROM_CATALOG is the recommended procedure for explaining static SQL after a package has been bound to the database and stored in the system catalogs.

Explaining a dynamic SQL statement after it is prepared

- Why would you want to do this?
  - Explaining an SQL statement in the present, may not produce the same access plan that was produced in the past.
  - The optimizer considers many environmental factors such as catalog statistics, the DB and DBM configuration and the compilation environment.
  - These factors might've changed since the statement was prepared and they might be hard to re-specify.
- Where might the prepared SQL statement exist?
  - Dynamic statement cache
  - System catalogs
  - Activity event monitor table
- The prepared SQL statement is called an *access section*
- The access section itself must be explained

Explain from Access Section

- Explain from access section? But didn't I just say not to use db2expln because it is based on the access section?
- Explain from access section looks at some additional information that is included with the real access section.
- It is also invoked as a stored procedure so it can be called via SQL



Explain from Access Section

- 3 step process:
  1. Obtain information identifying the location of the access section
  2. Use this information to call a stored procedure to write explain information to the explain tables
     - Four stored procedures, for different purposes
  3. Use an explain table formatting tool to view the explain
     - db2exfmt
     - Visual Explain
- Requires at least Db2 9.7 FP1

Explain from Access Section Routines (1|2)

- **EXPLAIN_FROM_SECTION**
  - Access section resides in:
    - Package cache
    - Package cache event monitor table
  - Identify access section by executable ID*and location ('P' or 'M')
- **EXPLAIN_FROM_ACTIVITY**
  - Access section resides in an activity event monitor table
  - Identify by application ID, activity ID, UOW ID, and activity event monitor name

* An opaque binary token that uniquely identifies the access section that was executed.

16

https://www.ibm.com/docs/en/db2/11.5?topic=facility-guidelines-capturing-section-explain-information

Explain from Access Section Routines (2|2)

- **EXPLAIN_FROM_CATALOG**
  - Access sections for static SQL
  - Identify by package name, package schema, unique ID, and section number
- **EXPLAIN_FROM_DATA**
  - Pass access section directly to procedure (BLOB type)
  - Point at access section in any arbitrary location
- **All procedures:**
  - Accept explain schema as input
  - Return explain table key as output

Explain from Access Section Example (1|2)

- Explaining a statement from the dynamic statement (package) cache

1. Locate the access section. Requires the **executable ID** .

```
SELECT SECTION_TYPE, EXECUTABLE_ID, VARCHAR(STMT_TEXT, 200) AS TEXT
    FROM TABLE( MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) as T
    WHERE T.NUM_EXEC_WITH_METRICS <> 0 AND STMT_TYPE_ID LIKE 'DML%'
    GROUP BY SECTION_TYPE, EXECUTABLE_ID, VARCHAR(STMT_TEXT, 200)
```

2. Call a stored procedure to write the explain information to the explain tables

```
CALL EXPLAIN_FROM_SECTION (
x'01000000000000005F000000000000000000000000202102111081356293590000', -- Executable ID
'P',                        -- Section is obtained from the package cache
NULL,                       -- Optional: package cache event monitor name
0,                          -- Member
NULL,                       -- Optional: explain table schema
?, ?, ?, ?, ? )             -- Output arguments used to locate entry in explain tables
```

Explain from Access Section Example (2|2)

- Output of EXPLAIN_FROM_SECTION call:

```
Value of output parameters
  --------------------------
  Parameter Name  : EXPLAIN_SCHEMA
  Parameter Value : DB2DOCS

  Parameter Name  : EXPLAIN_REQUESTER
  Parameter Value : DB2DOCS

  Parameter Name  : EXPLAIN_TIME
  Parameter Value : 2021-11-08-13.57.52.984001

  Parameter Name  : SOURCE_NAME
  Parameter Value : SQLC2H21

  Parameter Name  : SOURCE_SCHEMA
  Parameter Value : NULLID

  Parameter Name  : SOURCE_VERSION
  Parameter Value :
```

- Format the explain information:

```
db2exfmt -d gsdb -e db2docs -w 2021-11-08-13.57.52.984001 -n SQLC2H21 -s NULLID -t -#0
```

Explain from Access Section – Alternative Method

- Faster method when many statements need to be explained
- Collect section separately – explain later
  - Writing to the explain tables can be expensive
  - Can collect sections from one DB and explain on another!
- Available since Db2 10.5 FP4

Method 1	Method 2
1) Locate the executable ID for the access section	1) Locate the executable ID for the access section
2) Call EXPLAIN_FROM_SECTION to write explain information to explain tables	2a) Call MON_GET_SECTION table function to store access section in a user table
	2b) Call EXPLAIN_FROM_DATA to write explain information to explain tables
3) Format explain table contents (db2exfmt or visual explain)	3) Format explain table contents (db2exfmt or visual explain)

20

https://www.ibm.com/docs/en/db2/11.5?topic=mpf-mon-get-section-get-copy-section-from-package-cache

INSERT INTO REPOSITORY_SECTIONS(STMT_TEXT, STMTID, PLANID, SEMANTIC_ENV_ID, SECTION_DATA) SELECT STMT_TEXT, STMTID, PLANID, SEMANTIC_ENV_ID, (SELECT B.SECTION_ENV FROM TABLE(MON_GET_SECTION(A.EXECUTABLE_ID)) AS B) SECTION_DATA FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-2)) AS A

You can use this information to build a history of statements that ran. You can use the EXPLAIN_FROM_DATA stored procedure to examine the access plan for each saved statement by passing the saved section to the stored procedure.

Best method to explain static SQL statements (1|2)

- Note: access sections for static SQL are stored in the system catalogs
- There are 2 methods to explain static SQL:
  1. **Explain when the package is prepared or bound**
     - Must specify the EXPLAIN option on PREP or BIND command
       - YES = populate explain tables for static SQL at prepare/bind
       - ALL = same as YES, except dynamic SQL is explained at runtime
       - ONLY = explain only, but don't create a package
     - But what if EXPLAIN wasn't specified?
     - What if you didn't know there was actually a package, like there is for triggers and functions?

Best method to explain static SQL statements (2|2)

- There are 2 methods to explain static SQL:
  - **2. Explain based on the access section stored in the catalogs**
    - Use EXPLAIN_FROM_CATALOG stored procedure
    - Can get an explain after packages are bound, or triggers and functions are created
    - But there are a couple of gotchas:
      - Need to determine the package name for functions and triggers
        - (See example in speaker notes)
      - Can only explain 1 section at a time
      - A wrapper procedure or script can solve these problems

22

Example query to retrieve parameters for EXPLAIN_FROM_CATALOG call:

```
select r.routineschema, r.routinename, d.bschema as pkgschema, d.bname as pkgname,
s.version as pkgversion, s.sectno
from SYSCAT.STATEMENTS AS S,
SYSCAT.ROUTINEDEP AS D,
SYSCAT.ROUTINES   AS R
WHERE  D.BTYPE = 'K' AND
R.SPECIFICNAME = D.ROUTINENAME AND
R.ROUTINESCHEMA = D.ROUTINESCHEMA AND
S.PKGSCHEMA = D.BSCHEMA AND
S.PKGNAME = D.BNAME AND
R.ROUTINENAME = ;
```

```
>>-EXPLAIN_FROM_CATALOG----------------------------------------->
>--(--pkgschema--,--pkgname--,--pkgversion--,--sectno--,--explain_schema-->
>--,--explain_requester--,--explain_time--,--source_name--,--source_schema--,--
source_version--)-><
```

Collecting Access Sections

- Activity event monitors
  - Can be created for:
    - workloads
    - service classes
    - thresholds
    - work action
    - Specify COLLECT ACTIVITY DATA WITH SECTION
  - The Workload Management (WLM) feature is required to create these DB objects
  - Activity event monitors can be created for the default workloads and service classes shipped with Db2
    - SYSDEFAULTUSERWORKLOAD
      - SYSDEFAULTUSERCLASS
    - SYSDEFAULTADMWORKLOAD
    - But beware that ALL system activity will be monitored

Explain from Access Section

- Explain information is a subset of full explain
  - Full set of explain information is not included in the access section
  - Avoid increasing package cache memory
- The **most important** information is included
  - All operators, predicate text, estimated cardinality, most operator arguments
- What is missing:
  - Only TOTAL_COST and FIRST_ROW_COST are included (i.e. no CPU or I/O costs)
  - Stream column names
  - DB partition class columns
  - Many statistics
  - Implicitly (compiler) referenced objects
- But still contains MUCH more information than db2expln

How to verify the optimizer's estimates?

- Why would you want to do this?
- Consider what the optimizer does:
  - Models the execution of the access plan
  - **Estimates the number of rows** produced by each plan operator
  - Models the:
    - CPU, I/O, and memory costs for each plan operator
    - Communications costs (in partitioned and federated environments)
  - Selects the access plan with the cheapest cost
  - The estimate of the number of rows (AKA: cardinality) is **the most important factor** in the estimated cost
  - It is the first thing to verify when examining a suspect access plan

Explain with Actual Cardinality (1|5)

- Capture cardinality (number of rows) processed by each access plan operator at runtime
- Compare with the optimizer's estimates to identify possible access plan problems
  - Estimated cardinality is most important input to cost model
- Use explain from access section mechanism
- Fine print: doesn't collect actuals for column-organized processing

26

https://www.ibm.com/docs/en/db2/11.5?topic=information-capturing-accessing-section-actuals

https://www.ibm.com/docs/en/db2/11.5?topic=casa-obtaining-section-explain-actuals-investigate-poor-query-performance

Section actuals are runtime statistics collected during the execution of the section for an access plan. To capture a section with actuals, you use the activity event monitor. To access the section actuals, you perform a section explain using the EXPLAIN_FROM_ACTIVITY stored procedure.

To be able to view section actuals, you must perform a section explain on a section for which section actuals were captured (that is, both the section and the section actuals are the inputs to the explain facility). Information about enabling, capturing, and accessing section actuals is provided here.

Explain with Actual Cardinality (2|5)

- Must use activity event monitor
- 2 methods to activate:
  - Use WLM, specify COLLECT ACTIVITY DATA WITH DETAILS,SECTION
    - For workload, service class, threshold, work action
    - `ALTER WORKLOAD WL1 COLLECT ACTIVITY DATA WITH DETAILS,SECTION`
- OR
  - Use WLM_SET_CONN_ENV stored procedure to enable collection for the current connection

```
wlm_set_conn_env(null, '<collectactdata>with details, section
</collectactdata><collectsectionactuals>base</collectsectionactuals>');
```

- Use EXPLAIN_FROM_ACTIVITY stored procedure to populate explain tables

27

The mechanism for capturing a section, with section actuals, is the activity event monitor. An activity event monitor writes out details of an activity when the activity completes execution, if collection of activity information is enabled. Activity information collection is enabled using the COLLECT ACTIVITY DATA clause on a workload, service class, threshold, or work action. To specify collection of a section and actuals (if the latter is enabled), the SECTION option of the COLLECT ACTIVITY DATA clause is used. For example, the following statement indicates that any SQL statement, issued by a connection associated with the WL1 workload, will have information (including section and actuals) collected by any active activity event monitor when the statement completes:

ALTER WORKLOAD WL1 COLLECT ACTIVITY DATA WITH DETAILS,SECTION

Explain with Actual Cardinality (3|5)

- How to make it work:
  1. Enable a DB configuration parameter (not necessary if using WLM_SET_CONN_ENV)
     `section_actuals [base | none]`
  2. Create activity event monitor
  3. Create workload or use default workload (to collect activity data)
     - Alternative: WLM_SET_CONN_ENV for the current connection
  4. Activate activity event monitor
  5. Execute SQL statement(s), or stored procedures (more later)
  6. Locate SQL statement information in event monitor table to pass to EXPLAIN_FROM_ACTIVITY stored procedure
  7. Call EXPLAIN_FROM_ACTIVITY
  8. db2exfmt

28

Setup:

Enable DB configuration parameter if section actuals collection is to be enabled for the entire database.

Alternatively, they can be enabled at the session level using WLM_SET_CONN_ENV(). See steps further below.

DB2 UPDATE DATABASE CONFIGURATION USING SECTION_ACTUALS BASE;

Collection:

The easiest method is to use the db2caem tool:

db2caem -d <dbname> -st "SQL stmt"

Otherwise, these are the manual steps:

Create event monitor

CREATE EVENT MONITOR ACTEVMON FOR ACTIVITIES WRITE TO TABLE;

There are 2 methods to perform collection:

1) WLM setup:

Create workload or use default workload (to collect activity data)

2) Use WLM_SET_CONN_ENV stored procedure for the current connection

call wlm_set_conn_env(null, '<collectactdata>with details, section </collectactdata><collectsectionactuals>base</collectsectionactuals>');

Activate activity event monitor

SET EVENT MONITOR ACTEVMON STATE 1;

Execute SQL statement

Locate SQL statement information in event monitor table to pass to EXPLAIN_FROM_ACTIVITY stored procedure:

SELECT APPL_ID, UOW_ID, ACTIVITY_ID, STMT_TEXT FROM ACTIVITYSTMT_ACTEVMON;

-- APPL_ID UOW_ID ACTIVITY_ID STMT_TEXT

-- ------------------------- -------- -------------- ---------------

-- *N2.DB2INST1.0B5A12222841 1 1 SELECT * FROM ...

Populate the explain tables:

CALL EXPLAIN_FROM_ACTIVITY('*N2.DB2INST1.0B5A12222841', 1, 1, 'ACTEVMON', 'MYSCHEMA', ?, ?, ?, ?, ?);

Format the explain tables as usual e.g. db2exfmt

Explain with Actual Cardinality (4|5)

- A simpler option:

```
db2caem -d <dbname> -st "SQL stmt"
```

  - Db2 Capture Activity Event Monitor data tool
  - Automatically performs the steps shown on the previous page
  - New in Db2 11.5.6:
    - Supports schema and degree options

29

https://www.ibm.com/docs/en/db2/11.5?topic=commands-db2caem-capture-activity-event-monitor-data-tool

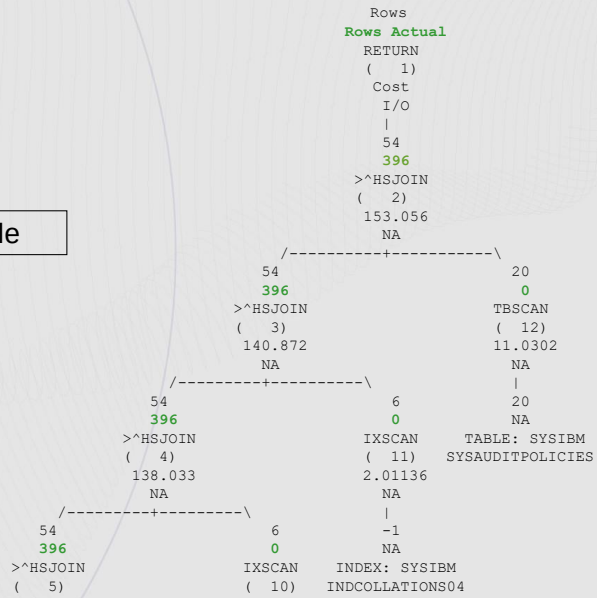
The db2caem tool automates the procedure of creating an activity event monitor.

•Run the db2caem command to create the activity event monitor to capture data for an SQL statement. This data can be collected with the db2support command. The information collected and generated by the db2caem tool includes: Detailed activity information captured by an activity event monitor including monitor metrics, for example total_cpu_time for statement execution

•Formatted EXPLAIN output, including section actuals (statistics for different operators in the access plan).

Explain with Actual Cardinality (5|5)

db2exfmt example



30

https://www.ibm.com/docs/en/db2/11.5?topic=information-analysis-section-actuals-in-explain-output

db2caem will produce partial results if the query doesn't run to completion. This can be very helpful for a query that takes too long to run and must be killed. See this technote for more details:

https://www.ibm.com/support/pages/node/279983

Explaining Stored Procedures (1|2)

- Use EXPLAIN_FROM_ACTIVITY method
- Follow the same steps used to collect explain with actual cardinality
  - Exclude **<collectsectionactuals>** option if actual cardinality is not required (but it is helpful to collect)
- Why? Because stored procedures sometimes:
  - reference user temporary (session) tables
  - dynamically generate SQL
- It is difficult to accurately capture and explain these statements outside procedure execution
- The activity event monitor captures EXACTLY what the procedure executed
- Not supported using db2caem

Explaining Stored Procedures (2|2)

- What if the stored procedure can't be run?
- Explain the procedure when it is created:

```
CALL SET_ROUTINE_OPTS('EXPLAIN YES')
CREATE PROCEDURE …
db2exfmt -d …
```

- Note: this will not explain dynamic SQL or SQL that references session tables

How to see the relevant statistics for a particular query

- Request the ***explain snapshot*** when collecting the explain
  - A binary object containing statistics information
  - Stored in EXPLAIN_STATEMENT.SNAPSHOT column (BLOB)
- Automatically formatted by db2exfmt
- Contains ALL relevant statistics
  - Not all statistics are included in EXPLAIN_OBJECT table

```
SET CURRENT EXPLAIN MODE EXPLAIN
SET CURRENT EXPLAIN SNAPSHOT EXPLAIN
<SQL stmt>
OR
EXPLAIN PLAN WITH SNAPSHOT FOR <SQL stmt>
```

- Statistics can also be formatted using EXPLAIN_FORMAT_STATS scalar function

33

EXPLAIN_FORMAT_STATS:

https://www.ibm.com/docs/en/db2/11.5?topic=routines-explain-format-stats-displays-statistics-information

```
SELECT EXPLAIN_FORMAT_STATS(SNAPSHOT)
  FROM EXPLAIN_STATEMENT
  WHERE EXPLAIN_REQUESTER = 'DB2USER1' AND
     EXPLAIN_TIME = timestamp('2021-05-12-14.38.11.109432') AND
     SOURCE_NAME = 'SQLC2F0A' AND
     SOURCE_SCHEMA = 'NULLID' AND
     SOURCE_VERSION = '' AND
     EXPLAIN_LEVEL = 'O' AND
     STMTNO = 1 AND
     SECTNO = 201
```

Collecting Relevant Statistics with Explain

- Why is this useful?
  - Obvious reason:
    - Helpful for understanding optimizer decisions
  - Non-obvious reason:
    - Statistics can change frequently due to automatic statistics collection
    - The statistics in the statistics cache could be more current than those in the system catalogs!
- I recommend always collecting the explain snapshot
- Not available with explain from access section
  - Storing all statistics along with the access section would consume too much package cache memory

34

https://www.ibm.com/docs/en/db2/11.5?topic=statistics-automatic-collection

Synchronous statistics collection does not store the statistics in the system catalog. Instead, the statistics are stored in a statistics cache and are later stored in the system catalog by an asynchronous operation. This storage sequence avoids the memory usage and possible lock contention that are involved in updating the system catalog. Statistics in the statistics cache are available for subsequent SQL compilation requests.

How to determine if an access plan has changed

- Straightforward approach is to compare the old and new explain table information
  - This is expensive, complex and some plan details don't have a significant impact on performance
- Solution: Access plans are uniquely identified by a ***plan id***
- The plan id is a 64-bit hash key based on important aspects of the access plan
  - Operator type, order, arguments, referenced objects
- Reported in explain output as a RETURN operator argument

```
PLANID  : (Access plan identifier)
    ad9edacfcc17d2e4
```

35

https://www.ibm.com/docs/en/db2/11.5?topic=reference-p#r0061351

The hash key value which identifies a query plan for a section.

The planid monitor element tracks important performance sensitive aspects of the access plan. Such aspects include the list and layout of access plan operators, identifiers of the objects that are being accessed, the number of each type of predicate for each operator, and performance sensitive operator arguments.

Access Plan IDs

- PLANID is returned by some monitoring functions
- PLANID is included in activity event monitor tables
  - ACTIVITYSTMT_<evmon_name>
- Allows tracking access plan changes for **current and historical** execution

Table Function Name	Description
MON_GET_ACTIVITY	Return a list of current activities (still executing).
MON_GET_ACTIVITY_DETAILS	Return information about a current activity as an XML document.
MON_GET_PKG_CACHE_STMT	Returns a point-in-time view of both static and dynamic SQL statements in the database package cache.
MON_GET_PKG_CACHE_STMT_DETAILS	Get package cache statement metrics as an XML document. Represents the accumulation of all metrics for statements in the package cache.
WLM_GET_WORKLOAD_OCCURENCE_ACTIVITIES	Same as MON_GET_ACTIVITY, but different authorizations.

Comparing Access Plans using PLANID

- PLANIDs are only comparable for a ***particular SQL statement*** executed in a ***particular semantic environment***
- Semantic environment id (SEMANTIC_ENV_ID)
  - Hash value computed over the **default schema** and **function path** values in the compilation environment
  - Needed to distinguish statements with identical text but executed with different schemas or function paths
- Statement id (STMTID)
  - Hash value computed over the **normalized** SQL statement text
  - Following statements have the same STMTID

```
SELECT NAME FROM CUSTOMER WHERE ID = 111
SELECT NAME FROM CUSTOMER WHERE ID = 222
```

  - There could be multiple PLANIDs for a given STMTID

37

https://www.ibm.com/docs/en/db2/11.5?topic=reference-s#r0061359

Use the semantic environment ID with the query statement ID monitor element (stmtid) to identify an SQL statement. The semantic compilation environment ID is used to distinguish queries that have the same statement text, but are semantically different because they reference different objects. For example, the table that is referenced in the statement SELECT * FROM T1 depends on the value of the default schema in the compilation environment. If two users with different default schemas issued this statement, there would be two entries for the statement in the package cache. The two entries would have the same stmtid value, but would have different values for semantic_env_id. The same applies to the function path, where 2 functions with the same name could have different implementations.

The rules for normalizing SQL statement text prior to computing STMTID are the same as for inexact matching of SQL statement text in optimization profiles.

The inexact statement matching rules are found here:

https://www.ibm.com/docs/en/db2/11.5?topic=matching-inexact

Comparing Access Plans using PLANID

- SEMANTIC_ENV_ID and STMTID are available wherever PLANID is available
- Included in explain tables:

```
SEMEVID : (Semantic environment identifier)
       0000000000000001
STMTID  : (Normalized statement identifier)
       fdc3021bec913ac9
```

- PLANID, STMTID and SEMANTIC_ENV_ID are stable across future fixpaks and releases
- PLANID does not change if DB objects (tables, indexes, etc.) are recreated with the same definition
  - PLANID is based on the **object name excluding schema**
  - Allows comparing PLANIDs across different DBs where tables with the same name exist in different schemas
  - **Useful for comparing access plans from a test and production system**

38

PLANIDs should be compared for the 'same' SQL statement, executed within the 'same' semantic or compilation environment. The semantic environment includes the schema and function path. For example, the same SQL statement could have different access plans if it were issued with different schemas, because it references tables with the same names, but different defintions. Monitoring and explain report a semantic environment id (SEMANTIC_ENV_ID) for this purpose. So the 'key' for a unique instance of an SQL statement is STMTID and SEMANTIC_ENV_ID.

The STMTID is a hash computed based on a normalized form of the original SQL statement text that strips out literals e.g. "select name from customer where id = 111" and ""select name from customer where id = 222" would have the same STMTID. This means that a given STMTID could have multiple PLANIDs associated with it, if the literals affected costing and plan selection. One can further differentiate the statements for a given STMTID by comparing the original SQL statement text. This approach allows the flexibility to track statements based only on STMTID (and SEMANTIC_ENV_ID) and PLANID, to reduce the amount of data collected, for applications that don't use input variables or statement concentration.

PLANID is based on DB object names excluding their schema. So the PLANID for “SELECT C1 FROM T1” could be the same for different T1s in different schemas. This is useful for comparing access plans when the T1s have the same definition but are in different schemas. This approach also prevents PLANIDs from changing if DB

objects are recreated.

Comparing Access Plans using PLANID

- These tips are about how to **avoid** using explain
- Comparing explains isn't necessary if the PLANID is the same
  - For the same STMTID and SEMANTIC_ENV_ID
- Even when access plans are captured using explain, only the PLANIDs need to be compared
  - Requires some SQL 'pivoting' to retrieve these IDs
  - See speaker notes for an example

39

```
select es.explain_time, varchar(statement_text,100),
(select varchar(ea.argument_value,20) as planid
from explain_argument ea
where ea.argument_type = 'PLANID' and
eo.operator_id = ea.operator_id and
es.explain_requester = ea.explain_requester and
es.explain_time = ea.explain_time),
(select varchar(ea.argument_value,20) as stmtid
from explain_argument ea
where ea.argument_type = 'STMTID' and
eo.operator_id = ea.operator_id and
es.explain_requester = ea.explain_requester and
es.explain_time = ea.explain_time),
(select varchar(ea.argument_value,20) as semantic_env_id
from explain_argument ea
where ea.argument_type = 'SEMEVID' and
eo.operator_id = ea.operator_id and
es.explain_requester = ea.explain_requester and
es.explain_time = ea.explain_time)
from explain_statement es, explain_operator eo
where
eo.operator_type = 'RETURN' and
es.explain_level = 'O' and
es.explain_requester = eo.explain_requester and
es.explain_time = eo.explain_time;
```

Explain Diagnostic Messages

- Explain can provide helpful information such as:
  - Notification about missing statistics
  - Information about whether or not materialized query tables (MQTs) or statistical views could be matched
  - Syntax errors when using optimization profiles
  - Index recommendations in limited contexts (for zig-zag join)
  - More will be added in future releases

Explain Diagnostic Messages

- 2 additional explain tables:
  - EXPLAIN_DIAGNOSTICS and EXPLAIN_DIAGNOSTICS_DATA
- Formatted message text appears in the db2exfmt output or visual explain:

```
EXP0020W Table has no statistics. The table "DB2DBA"."SALES" has not had runstats run on it. This may
  result in a sub-optimal access plan and poor performance.

EXP0060W The following materialized query table (MQT) or statistical view was not eligible for query
  optimization: "DB2DBA"."SV_STORE". The MQT cannot be used for query optimization because one or more
  tables, views or subqueries specified in the MQT could not be found in the query that is being
  explained.

EXP0147W  The following statistical views may have been used by the optimizer to estimate cardinalities:
"DB2DBA"."SV_STORE".

EXP0256I Analysis of the query shows that the query might execute faster if an additional index was
created to enable zigzag join. Schema name: "DB2DBA". Table name: "SALES". Column list:
"PERIODKEY,REGIONKEY,PRODUCTKEY".
```

Upgrading Explain Tables

- Sometimes new Db2 releases add new explain tables and/or explain table columns
- Previous explain tables are always upward compatible
  - But some new functionality is skipped if tables or columns are missing
- Solution – upgrade explain tables using:
  - db2exmig
  - OR
  - SYSINSTALLOBJECTS stored procedure

```
CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN', 'M',
CAST (NULL AS VARCHAR(128)), -- Table space name. Table space of existing
                             -- explain tables is always used.
CAST (NULL AS VARCHAR(128))) -- Optional schema. Default is SYSTOOLS.
```

42

https://www.ibm.com/docs/en/db2/11.5?topic=tasks-upgrading-explain-tables

Why are my explain tables growing?

- Db2 only inserts into the explain tables
- Allows more flexibility
- But deleting everything is easy to do:

```
DELETE FROM EXPLAIN_INSTANCE
```

- Referential integrity is defined by default and EXPLAIN_INSTANCE is at the top of the hierarchy