

IDUG

2026

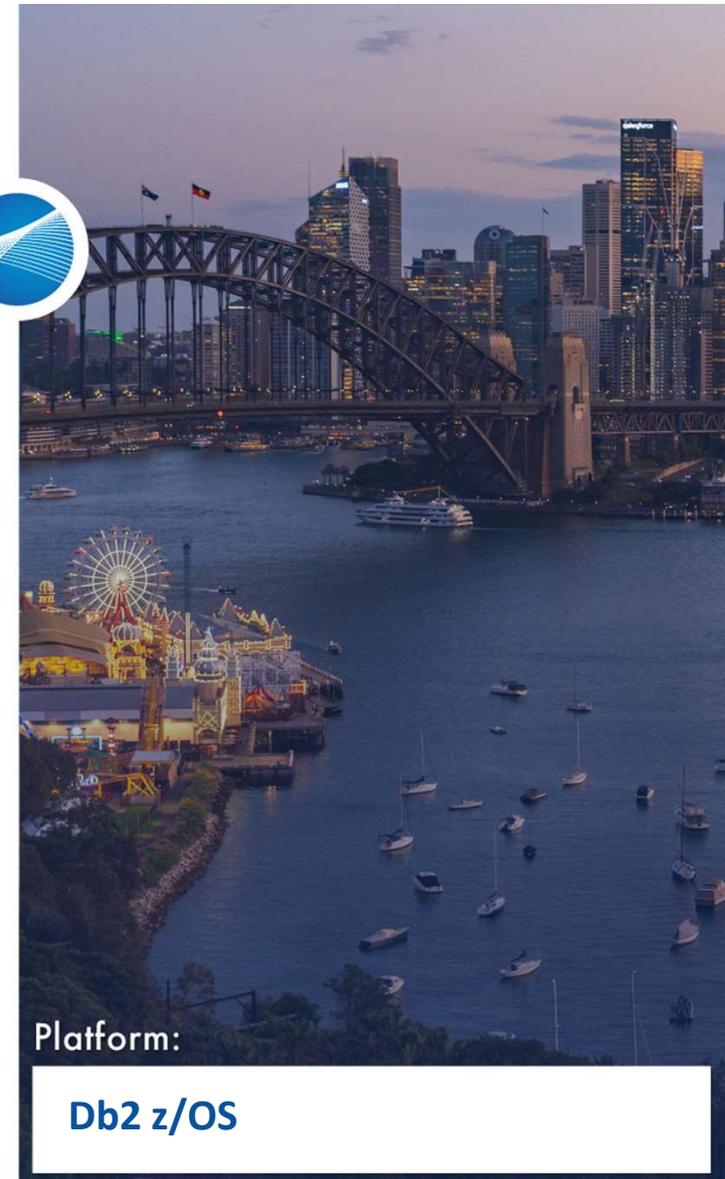
Sydney | March 16 - 18

AU Db2 TECH CONFERENCE

Exploring Db2 Explain Facility for the next Gen of Db2
DBAs and using AI to bridge the skills gap

Phill_Baker@bmc.com

Session Code: A12



Platform:

Db2 z/OS

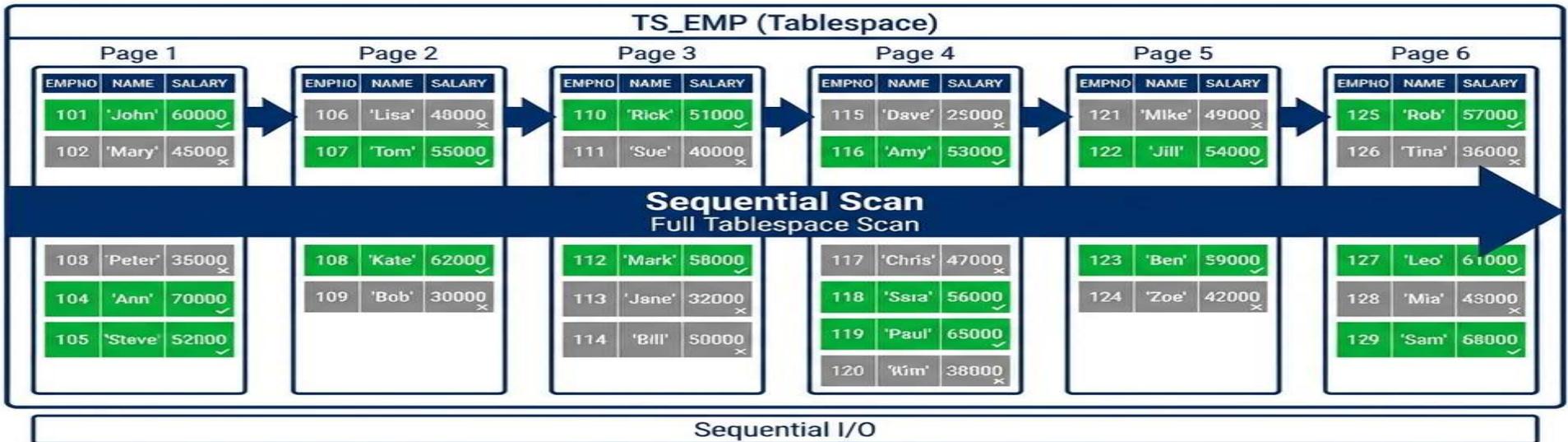
AGENDA

- Db2 access paths
- Filter factors
- Db2 explain facility
- Using AI with Db2 explain
- Influencing access paths

The access paths

Tablespace scan (accesstype=R)

```
SELECT * FROM EMP WHERE SALARY > 50000
```



NOTE: All pages read, only matching rows returned



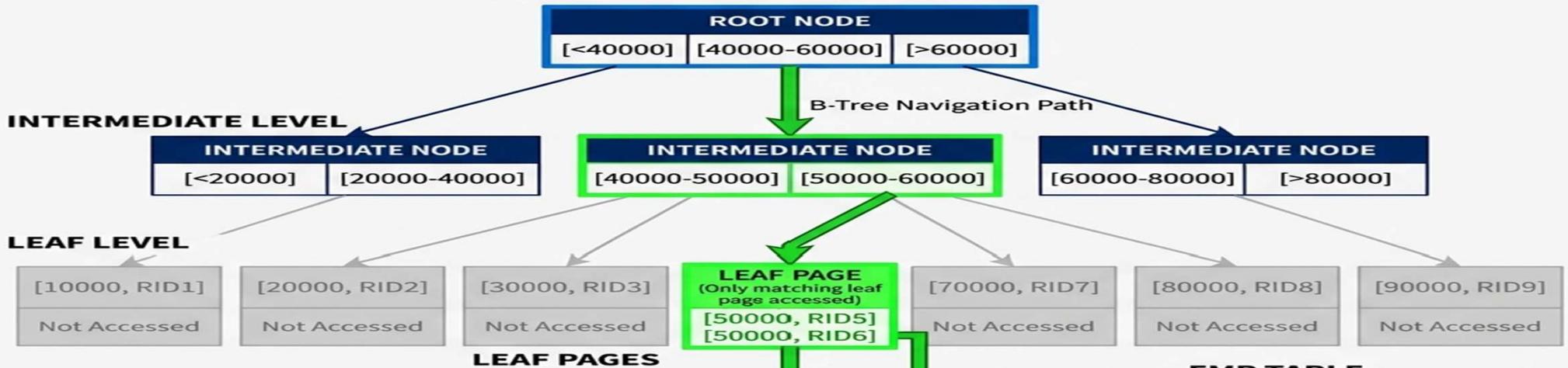
TS SCAN ACESSTYPE=R

PLANNO	TNAME	ACESSTYPE	MATCHCOLS	ACCESSNAME	METHOD	PREFETCH
1	EMP	R	0		0	S

Db2 z/OS Matching Index Scan

SELECT * FROM EMP WHERE SALARY = 50000

IX_EMP_SALARY INDEX B-Tree Structure



EMP TABLE

EMP_ID	EMP_NAME	SALARY	RID
E001	John Doe	10000	RID1
E002	Jane Smith	20000	RID2
E003	Bob Johnson	30000	RID3
E004	Alice Williams	40000	RID4
E005	Michael Brown	50000	RID5
E006	Emily Davis	50000	RID6
E007	Chris Martin	60000	RID7
E008	Patricia Taylor	70000	RID8
E009	Robert Clark	80000	RID9

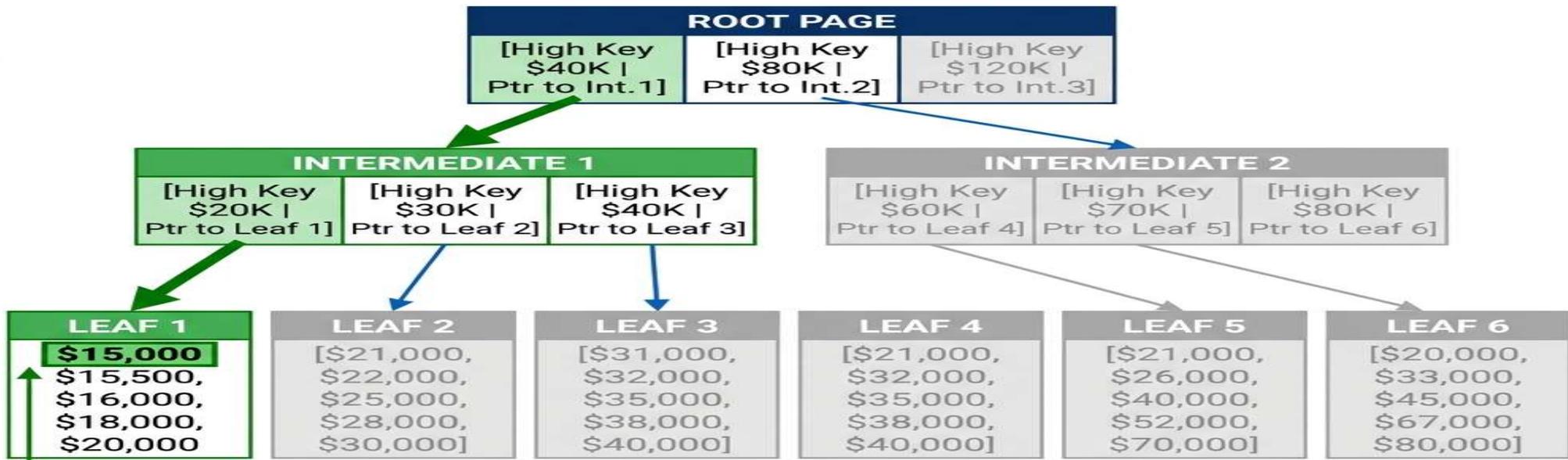
accesstype=l

Matching index scan **ACCESSTYPE=I** and **MATCHCOLS >=1**

PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	METHOD	PREFETCH
1	EMP	I	1	IX_EMP_DEPT	0	

Db2 z/OS One-Fetch Index Access (I1)

SQL: SELECT MIN(SALARY) FROM EMP



Absolute First Entry

LABELS

- ACESSTYPE = I1 (One-Fetch)
- Single Probe Efficiency

NOTE: Data values strictly increase across all pages.
Db2 probes directly to the first logical leaf entry to satisfy MIN query with ONE fetch.

Access paths

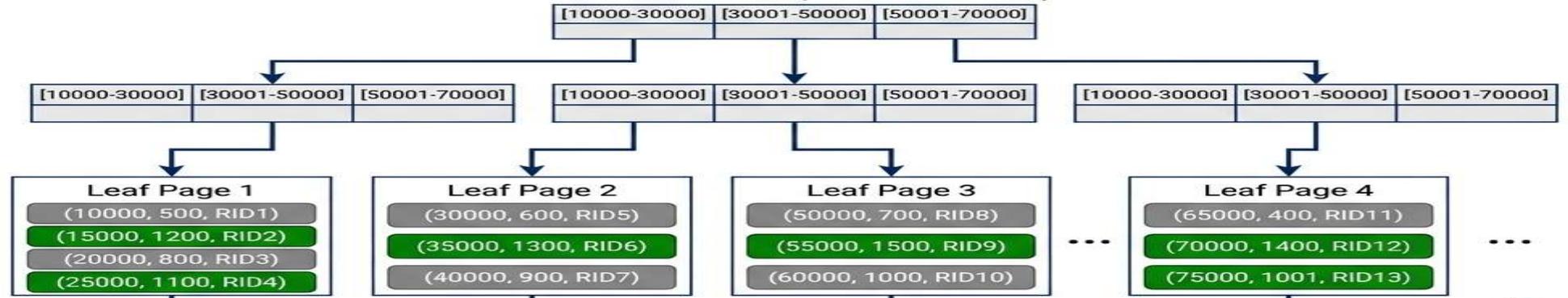
PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	METHOD
1	EMP	I1	0	IX_EMP_SALARY	0

Db2 z/OS Non-Matching Index Scan (MATCHCOLS=0)

SELECT * FROM EMP WHERE BONUS > 1000

Index IX_EMP_SALARY exists on (SALARY, BONUS) - leading column not used

IX_EMP_SALARY (SALARY, BONUS)



Scans ALL index leaf pages →

EMPNO	ENAME	SALARY	BONUS	RID
101	'SMITH'	15000	1200	RID2

EMPNO	ENAME	SALARY	BONUS	RID
101	'SMITH'	15000	1200	RID2
102	'JONES'	35000	1300	RID6
103	'BROWN'	55000	1500	RID9
104	'DAVIS'	70000	1400	RID12
...

Non-Matching Index Scan

MATCHCOLS = 0

Index used but no matching columns
- full leaf scan required

! Less efficient than a matching scan - processes every leaf page regardless of predicate selectivity.

Access paths



PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	PREFETCH
1	EMP	I	0	IX_EMP_SALARY_BONUS	S

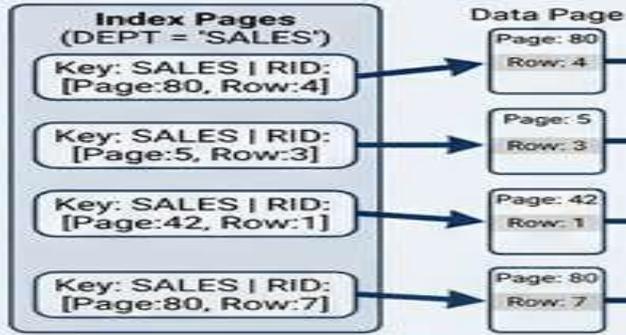
Db2 z/OS List Prefetch Mechanism

SQL Query: `SELECT * FROM EMP WHERE DEPT = 'SALES'` (using non-clustering index)

Phase 1: RID EXTRACTION

PREFETCH = L (List Prefetch)

Non-Clustering Index



Phase 2: RID SORT

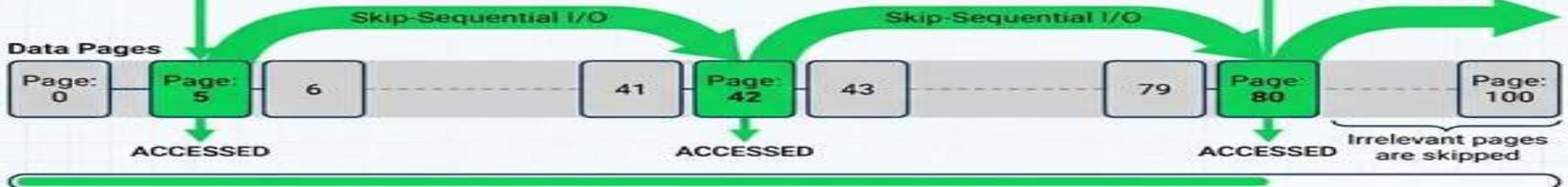
RID Pool Sorting Phase



RIDs sorted by Page Number

Phase 3: LIST PREFETCH

Efficient Skip-Sequential I/O



Access paths



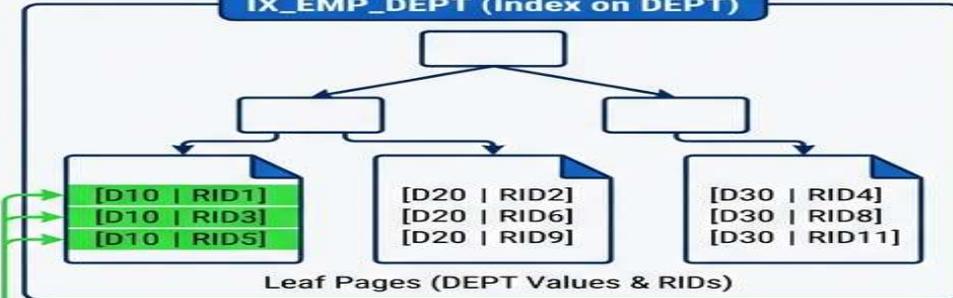
PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	PREFETCH
1	EMP	I	1	IX_EMP_DEPT	L

Multi-Index Access

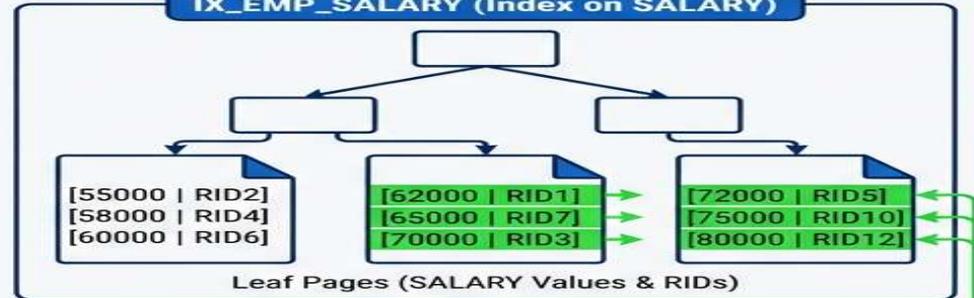
Db2 z/OS Multi-Index Access (Multiple Index Access)

`SELECT * FROM EMP WHERE DEPT = 'D10' AND SALARY > 60000`

IX_EMP_DEPT (Index on DEPT)



IX_EMP_SALARY (Index on SALARY)



RID Pool

RID Intersection Operation (AND Logic)

RIDs from
IX_EMP_DEPT
(DEPT='D10')



RIDs from
IX_EMP_SALARY
(SALARY>60000)

RID List Intersection: Combines multiple indexes

EMP Table (Data Rows)

RID	EMP_ID	NAME	DEPT	SALARY	...
RID2	E102	Jane Roe	D20	55000	...
RID4	E104	Mike Brown	D30	58000	...
RID1	E101	John Doe	D10	62000	...
RID3	E103	Alice Smith	D10	70000	...
RID5	E105	Bob Johnson	D10	72000	...
RID7	E107	Sara Davis	D20	65000	...

Qualifying Rows Matching **BOTH** Conditions

Note: Efficient when each index is highly selective

Access paths

PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	PREFETCH	MIXOPSEQ
1	EMP	M	0		L	0
1	EMP	MX	1	IX_EMP_DEPT		1
1	EMP	MX	1	IX_EMP_SALARY		2
1	EMP	MI	0			3

Db2 z/OS Nested Loop Join

```
SELECT * FROM DEPT D INNER JOIN EMP E ON D.DEPTNO = E.DEPTNO
```

Outer Table (DEPT)

DEPTNO	DNAME
D10	Sales
D20	Marketing
D30	IT

Step 1 - Read first row from DEPT (D10)

IX_EMP_DEPTNO
(Index on DEPTNO)

Inner Table (EMP)

EMPNO	ENAME	DEPTNO
E101	Alice	D10
E102	Bob	D10
E103	Charlie	D20
E104	David	D20
E105	Eve	D10
E106	Frank	D30
E107	Grace	D20

Step 2 - Scan EMP table for matching DEPTNO=D10 rows (use index IX_EMP_DEPTNO)

Step 3 - Return matched pairs

Step 4 - Repeat for next DEPT row

(D10, Sales, E101, Alice, D10)
(D10, Sales, E102, Bob, D10)
(D10, Sales, E105, Eve, D10)

Note: Efficient when outer table is small and inner table has index

Access paths

PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	METHOD	PREFETCH
1	DEPT	R	0		0	S
2	EMP	I	1	IX_EMP_DEPTNO	1	

Db2 z/OS Merge Scan Join (Sort Merge Join)

`SELECT * FROM DEPT D INNER JOIN EMP E ON D.DEPTNO = E.DEPTNO`

Phase 1

Sort Phase (DEPT)

DEPT Table

DEPTNO	DNAME	LOC
D10	ACCOUNTING	NEW YORK
D20	RESEARCH	DALLAS
D30	SALES	CHICAGO
D40	OPERATIONS	BOSTON

Sort by DEPTNO
(or Index Scan)

Sorted DEPT

DEPTNO	DNAME	LOC
D10	ACCOUNTING	NEW YORK
D20	RESEARCH	DALLAS
D30	SALES	CHICAGO

Sorted Input 1

Phase 2

Sort Phase (EMP)

EMP Table

EMPNO	ENAME	DEPTNO
7839	KING	D10
7698	BLAKE	D30
7782	CLARK	D10
7566	JONES	D20
7902	FORD	D20
7369	SMITH	D20
7934	MILLER	D30

Sort by
DEPTNO

Sorted EMP

EMPNO	ENAME	DEPTNO
7782	CLARK	D10
7839	KING	D10
7566	JONES	D20
7369	SMITH	D20
7902	FORD	D20
7698	BLAKE	D30
7934	MILLER	D30

Sorted Input 2

Phase 3

**MERGE PHASE
&
PARALLEL SCAN**

DEPTNO	DNAME	LOC
D10	ACCOUN	D10
D20	RESEAR	D20
D30	SALES	D20
D30	SALES	D30
D40		

Scan Pointer 1
(DEPT)

EMPNO	ENAME	DEPTNO
7782	CLARK	D10
7839	KING	D10
7566	JONES	D20
7369	SMITH	D20
7902	FORD	D20
7698	BLAKE	D30
7934	MILLER	D30

Scan Pointer 2
(EMP)

Merge Scan
Join Operation

DEPTNO	DNAME	LOC	EMPNO	ENAME	DEPTNO
D10	ACCOUNTING	NEW YORK	7782	CLARK	D10
D10	ACCOUNTING	NEW YORK	7839	KING	D10
D20	RESEARCH	DALLAS	7566	JONES	D20
D20	RESEARCH	DALLAS	7369	SMITH	D20
D20	RESEARCH	DALLAS	7902	FORD	D20
D30	SALES	CHICAGO	7698	BLAKE	D30
D30	SALES	CHICAGO	7934	MILLER	D30

JOIN RESULT SET

Both inputs sorted on join key

Efficient for large tables - single pass through sorted data

Access paths



PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	ACCESSNAME	METHOD	SORTN_JOIN	SORTC_JOIN
1	DEPT	R	0		0	Y	N
2	EMP	R	0		2	N	Y

Filter factors

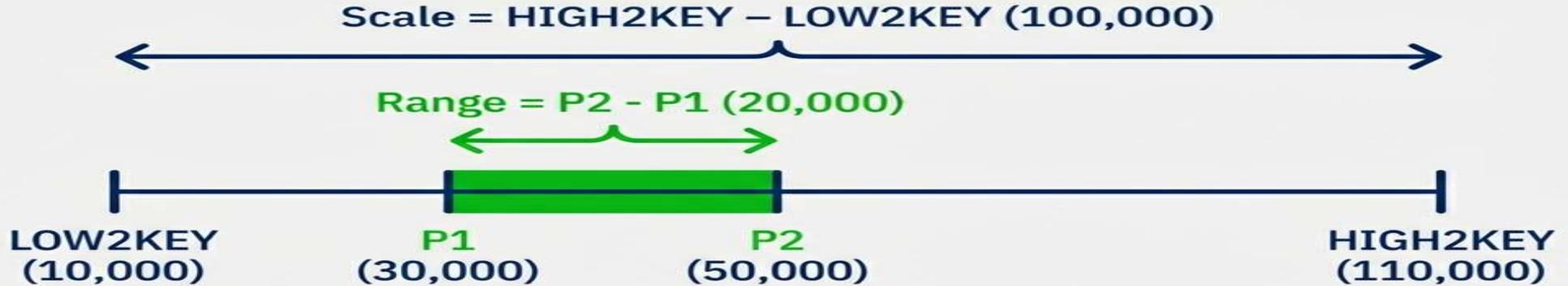
Default filter factor

Predicate Type	Filter Factor
Col = <i>constant</i>	0.04
Col <> <i>constant</i>	0.96
Col IS NULL	0.04
Col IS NOT DISTINCT FROM	0.04
Col IS DISTINCT FROM	0.96
Col IN (<i>constant-list</i>)	MIN(<i>number-of-constants</i> /25, 1.0)
Col <i>Op constant</i> , where <i>Op</i> is one of these operators: <, <=, >, >=.	0.33
Col LIKE <i>constant</i>	0.10
Col BETWEEN <i>constant1</i> and <i>constant2</i>	0.10

filter factors for uniformed data and positive COLCARDF

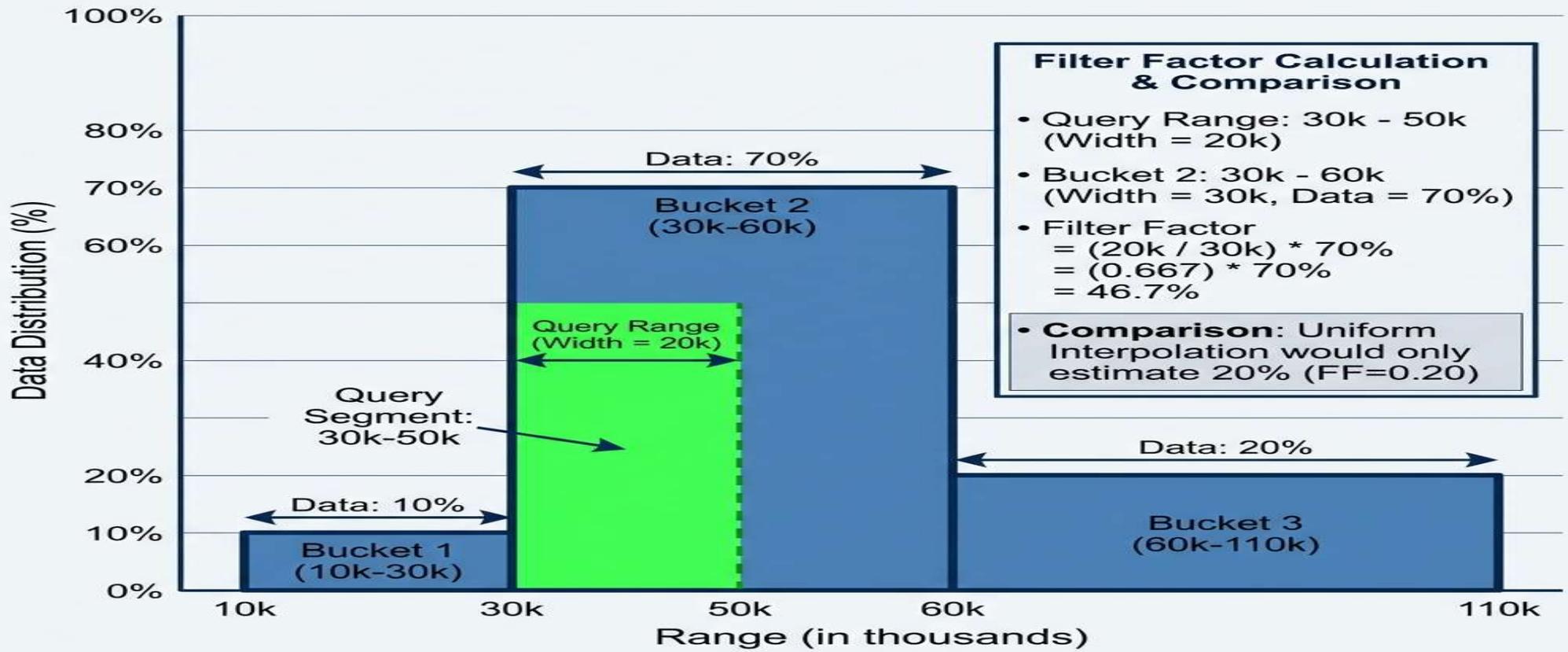
Predicate type	Filter factor
Col = <i>constant</i>	$1/\text{COLCARDF}$
Col <> <i>constant</i>	$1 - (1/\text{COLCARDF})$
Col IS NULL	$1/\text{COLCARDF}$
Col IS NOT DISTINCT FROM	$1/\text{COLCARDF}$
Col IS DISTINCT FROM	$1 - (1/\text{COLCARDF})$
Col IN (constant list)	<i>number of constants</i> / COLCARDF
Col Op1 ¹ constant	interpolation formula
Col Op2 ² constant	interpolation formula
Col LIKE constant	interpolation formula
Col BETWEEN <i>constant1</i> and <i>constant2</i>	interpolation formula

Interpolation Formula selectivity



$$\begin{aligned}
 \text{Filter Factor} &= \text{Range} / \text{Scale} \\
 &= 20,000 / 100,000 \\
 &= 0.20
 \end{aligned}$$

Db2 z/OS Histogram Statistics with Filter Factor Calculation for Specific Range



Explain facility

[PLAN_TABLE](#) contains information about access paths that is collected from the results of EXPLAIN statements.

[DSN_COLDIST_TABLE](#) contains non-uniform column group statistics that are obtained dynamically by Db2 from non-index leaf pages.

[DSN_DETCOST_TABLE](#) contains information about detailed cost estimation of the mini-plans in a query.

[DSN_FILTER_TABLE](#) contains information about how predicates are used during query processing.

[DSN_FUNCTION_TABLE](#) contains descriptions of functions that are used in specified SQL statements.

[DSN_KEYTGTDIST_TABLE](#) contains non-uniform index expression statistic that are obtained dynamically by the Db2 optimizer.

[DSN_PGRANGE_TABLE](#) contains information about qualified partitions for all page range scans in a query.

[DSN_PGROUP_TABLE](#) contains information about the parallel groups in a query.

[DSN_PREDICAT_TABLE](#) contains information about all of the predicates in a query. It is also used as input when you issue a BIND QUERY command to override the selectivity of predicates for matching SQL statements.

[DSN_PREDICATE_SELECTIVITY](#) contains information about the selectivity of predicates that are used for access path selection. It is used as an input table for the BIND QUERY command when selectivity overrides are specified.

[DSN_PTASK_TABLE](#) contains information about all of the parallel tasks in a query.

[DSN_QUERYINFO_TABLE](#) contains information about the eligibility of query blocks for automatic query rewrite, information about the materialized query tables that are considered for eligible query blocks, reasons why ineligible query blocks are not eligible, and information about acceleration of query blocks.

[DSN_QUERY_TABLE](#) contains information about a SQL statement, and displays the statement before and after query transformation.

[DSN_SORTKEY_TABLE](#) contains information about sort keys for all of the sorts required by a query.

[DSN_SORT_TABLE](#) contains information about the sort operations required by a query.

[DSN_STATEMENT_CACHE_TABLE](#) contains information about the SQL statements in the statement cache, information captured as the results of an EXPLAIN STATEMENT CACHE ALL statement.

[DSN_STATEMNT_TABLE](#) contains information about the estimated cost of specified SQL statements.

[DSN_STAT_FEEDBACK](#) contains recommendations for capturing missing or conflicting statistics that are defined during EXPLAIN.

[DSN_STRUCT_TABLE](#) contains information about all of the query blocks in a query.

[DSN_VIEWREF_TABLE](#) contains information about all of the views and materialized query tables that are used to process a query.

```
EXPLAIN PLAN SET QUERYNO = 1 FOR
SELECT * FROM SYSIBM.SYSTABLES T JOIN SYSIBM.SYSCOPY C
ON T.DBNAME = C.DBNAME AND T.TSNAME = C.TSNAME
where t.dbname in (select name from
sysibm.sysdatabase) and t.name like 'Phill%'
order by 1
EXPLAIN STMTCACHE ALL;

EXPLAIN STMTCACHE STMTID 4521;

REBIND PACKAGE (MYCOLL.MYPGM) EXPLAIN (YES)
```

PLAN_TABLE

QUERYNO ↕	QBLOCKNO ↕	PLANNO ↕	METHOD ↕	TNAME ↕	ACCESSTYPE ↕	MATCHCOLS ↕	ACCESSNAME ↕	INDEXONLY ↕	SORTS ↕	PREFETCH ↕
1	1	1	0	SYSTABLES	I	0	DSNDTX01	N	NNNNNNNN	D
1	1	2	1	DSNWFQB(02)	O	0		N	NNNNNNNN	
1	1	3	1	SYSCOPY	I	2	DSNUCH01	N	NNNNNNNN	
1	1	4	3			0		N	NNNNNNYN	
1	2	1	0	SYSDATABASE	I	1	DSNDDH01	Y	NNNNNNNN	

DSN_STATEMENT_TABLE

QUERYNO	STMT_TYPE	COST_CATEGORY	PROCMS	PROCSU	TOTAL_COST	REASON
1	SELECT	A	3	132	48.41550076555596	

DSN_PREDICATE_TABLE

QUERYNO	QBLOCKNO	PREDNO	TYPE	FILTER_FACTOR	SEARCHARG	TEXT	JOIN	AFTER_JOIN	ADDED_PRED
1	1	1	AND	1.0000000000000000	Y	(CAST(CAST("HAUPXB0"."DSNWFQB(02)". "NAME" AS ...	N		
1	1	2	IN	1.0000000000000000	N	"T"."DBNAME" IN (SELECT "SYSIBM"."SYSDATABASE". "...	N		
1	1	3	LIKE	0.0003324467688799	Y	"T"."NAME" LIKE 'Phill%'	N		
1	1	4	EQUAL	0.0015797787345946	Y	CAST(CAST("T"."DBNAME" AS VARCHAR(24)) AS CHAR(...	Y		
1	1	5	EQUAL	0.0003720237873495	Y	CAST(CAST("T"."TSNAME" AS VARCHAR(24)) AS CHAR(8...	Y		
1	1	6	EQUAL	0.0004608293529600	Y	"T"."DBNAME"="HAUPXB0"."DSNWFQB(02)". "NAME"	Y		
1	1	8	EQUAL	0.0004608293529600	Y	CAST(CAST("HAUPXB0"."DSNWFQB(02)". "NAME" AS V...	N		T
1	1	9	COMPOUND	0.0000882122258190	Y	COMPOUND: 3,6			
1	1	10	COMPOUND	0.0002079002151731	Y	COMPOUND: 4,5			
1	1	11	COMPOUND	0.0004608293529600	Y	COMPOUND: 6,8			
1	1	12	COMPOUND	0.0000881466694409	Y	COMPOUND: 3,4,5			
1	2	7	EQUAL	0.0004608293529600	Y	"T"."DBNAME"="SYSIBM"."SYSDATABASE". "NAME"	N		C

<https://www.ibm.com/docs/en/db2-for-zos/13.0.0?topic=efficiently-summary-predicate-processing>

Indexable and stage 1 predicates ³¹

The following predicates might be evaluated by matching index access, during index screening, or after data page access during stage 1 processing. COL = value ^{15, 31}

COL = noncol expr ^{9, 11, 12, 13, 15, 29, 31, 32}
 COL IS NULL ^{20, 21}
 COL op value ^{13, 31}
 COL op noncol expr ^{9, 11, 12, 13, 29, 31, 32}
 value BETWEEN COL1 AND COL2 ^{13, 32}
 COL BETWEEN value1 AND value2 ¹³
 COL BETWEEN noncol expr 1 AND noncol expr 2 ^{9, 11, 12, 13, 23, 29}
 COL BETWEEN expr-1 AND expr-2 ^{5, 7, 11, 12, 13, 14, 15, 27, 29}
 COL LIKE 'pattern' ²⁹
 COL IN (list) ^{17, 18}
 COL IS NOT NULL ²¹
 COL LIKE host variable ^{2, 29}
 COL LIKE UPPER ('pattern') ²⁹
 COL LIKE UPPER (host-variable) ^{2, 29}
 COL LIKE UPPER (SQL-variable) ^{2, 29}
 COL LIKE UPPER (global-variable) ^{2, 29}
 COL LIKE UPPER (CAST ('pattern' AS data-type)) ^{2, 29}
 COL LIKE UPPER (CAST (host-variable AS data-type)) ^{2, 29}
 COL LIKE UPPER (CAST (SQL-variable AS data-type)) ^{2, 29}
 COL LIKE UPPER (CAST (global-variable AS data-type)) ^{2, 29}
 T1.COL = T2.COL
 T1.COL op T2.COL
 T1.COL = T2 col expr ^{5, 9, 11, 12, 14, 15, 25, 27, 29}
 T1.COL op T2 col expr ^{5, 9, 11, 12, 13, 14, 15, 29}
 COL = (noncor subq) ²⁸
 COL op (noncor subq) ^{22, 29}
 COL = ANY (noncor subq) ^{22, 29}
 (COL1,...COLn) IN (noncor subq) ²⁹
 COL = ANY (cor subq) ^{19, 22, 29}
 COL IS NOT DISTINCT FROM value ¹⁶
 COL IS NOT DISTINCT FROM noncol expr ^{9, 11, 12, 15, 29}
 T1.COL1 IS NOT DISTINCT FROM T2.COL2 ^{3, 4}
 T1.COL1 IS NOT DISTINCT FROM T2.col expr ^{5, 9, 11, 12, 14, 15, 29}
 COL IS NOT DISTINCT FROM (noncor subq)
 SUBSTR(COL,1,n) = value
 SUBSTR(COL,1,n) op value
 DATE(COL) = value ³³
 DATE(COL) op value ³³
 YEAR(COL) = value ³³
 YEAR(COL) op value ³³

Stage 1 not indexable predicates ³¹

The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access. COL <> value ^{8, 11}

COL <> noncol expr ^{8, 11, 29}
 COL NOT BETWEEN value1 AND value2
 COL NOT IN (list)
 COL NOT LIKE 'char' ²⁹
 COL LIKE '%char' ^{1, 29}
 COL LIKE '_char' ^{1, 29}
 T1.COL <> T2 col expr ^{8, 11, 27, 29}
 COL op ANY (noncor subq) ²²
 COL op ALL (noncor subq)
 COL IS DISTINCT FROM value ^{8, 11}
 COL IS DISTINCT FROM (noncor subq)

Stage 2 predicates

The following predicates must be processed during stage 2, after the data is returned. COL

BETWEEN COL1 AND COL2 ¹⁰
 value NOT BETWEEN COL1 AND COL2
 value BETWEEN col expr and col expr ³²
 T1.COL <> T2.COL
 T1.COL1 = T1.COL2 ^{3, 25}
 T1.COL1 op T1.COL2 ³
 T1.COL1 <> T1.COL2 ³
 COL = ALL (noncor subq)
 COL <> (noncor subq) ²²
 COL <> ALL (noncor subq)
 COL NOT IN (noncor subq)
 COL = (cor subq) ⁵
 COL = ALL (cor subq)
 COL op (cor subq) ³
 COL op ANY (cor subq) ²²
 COL op ALL (cor subq)
 COL <> (cor subq) ³
 COL <> ANY (cor subq) ¹⁹
 (COL1,...COLn) IN (cor subq)
 COL NOT IN (cor subq)
 (COL1,...COLn) NOT IN (cor subq)
 T1.COL1 IS DISTINCT FROM T2.COL2 ³
 T1.COL1 IS DISTINCT FROM T2.col expr ^{8, 11}
 COL IS NOT DISTINCT FROM (cor subq)
 EXISTS (subq) ¹⁹
 expression = value ^{27, 32}
 expression <> value ²⁷
 expression op value ^{27, 32}
 expression op (subq)
 NOT XML EXISTS
 CASE expression WHEN expression ELSE expression END = value ³²

Indexable but not stage 1 predicates

The following predicates can be processed during index access, but cannot be processed during stage 1. XML EXISTS ²⁶

Sargable

Feature	Stage 1 (Sargable)	Stage 2 (Non-Sargable)
Evaluated By	Data Manager (DM)	Relational Data Services (RDS)
Performance	Very Good - Filters data early	Poor - Filters data late
Data Movement	Only matching rows are passed to RDS	All rows are passed to RDS, then filtered
Index Usage	Can often use an index (Indexable)	Can never use an index to limit the scan

The term **Sargable** is actually an acronym created by researchers at IBM in the late 1970s. It stands for Search ARGument ABLE.

So, when a DBA says a query isn't "Sargable," they just mean the database engine isn't *able* to use that predicate as a direct *search argument* against an index!

Using AI with Db2 explain

Everything's better with AI

You are a Db2 z/OS DBA. You have VS Code install on windows laptop. You have IBM Db2 Developer extension for vs code. This is configured and can run SQL.

I have Db2 v11.5.4_jdbc_sqlj.tar located in
C:\Users\OneDrive\Downloads this has jdbc files for Db2 connection.

I have Db2 license file db2consv_ee.lic located in
C:\Users\OneDrive\Downloads.

I want you to code me a VS CODE plugin that can explain Db2 SQL, read the Db2 catalog tables and make recommendations on SQL performance and missing Db2 indexes

Connection Failed: Missing z/OS License

The error confirms that the code is reaching your mainframe, but the Db2 driver is rejecting the connection because it's missing the **z/OS-specific license JAR**.

You provided `db2consv_ee.lic`, which is typically for Db2 *Server* (LUW), but connecting to **Db2 z/OS** requires a specific file named: `db2jcc_license_cisuz.jar`

Action Required: Do you have this file (`db2jcc_license_cisuz.jar`)? It is usually found in the license folder of your Db2 Connect installation or provided by your mainframe admin.

If you have it, please tell me the full path to it, and I will copy it into the extensions lib folder and fix the issue.

Debugging Empty Explain Plan

Resolved empty plan table issue by defaulting schema to authenticated user instead of SYSIBM. Re-built and re-installed.

Debugging SQL Column Error

Confirmed CLUSTRATIO is invalid column in modern Db2 z/OS (should be CLUSTERRATIO). Updating CatalogService.ts. Also ensuring

models.ts reflects this change. Then rebuilding.

Fixed: "CLUSTRATIO" Error (-206)

You are absolutely right—this is for **Db2 z/OS**, and the catalog column is named **CLUSTERRATIO**, not **CLUSTRATIO** (which I used by mistake).

I'm sorry. Without seeing the new error, I'm flying blind.

Please paste the latest error message. (It will help me decide if I need to try gemini-pro, check your API key format, or fix the JSON payload).

Debugging Persistent Failure

User doubts usage is the issue. Creating a standalone debug-gemini.js script to manually invoke the API (with the exact same code logic as the extension) to isolate if the issue is environmental/code-based or truly a platform return. This will bypass the extension UI entirely.

Restoration Complete!

I apologize for the scare! The reason it disappeared from your Command Palette is that I successfully **Uninstalled** the old one, but the **Install** of the new one stalled waiting for a confirmation prompt.

I have just finished the installation of **v0.0.3**.

Please Try Again:

Using AI with Db2 explain

The screenshot shows an IDE interface with a dark theme. The top menu bar includes File, Edit, Selection, View, and a search bar for 'Untitled (Workspace)'. The left sidebar contains several panels: 'DB2 DEVELOPER EX...', 'DB2 FOR Z/OS CONNECTI...', 'QUERY HISTORY' (listing test_query.sql and Untitled-1), 'SQL TUNING SERVICES SE...', and 'TUNING HISTORY'. The main editor area displays a SQL query in a file named 'Untitled-1':

```

1 explain plan SET QUERYNO = 1 FOR select
2
3 select * from sysibm.systables where name like 'Phill%'
4
5 what time is it
  
```

Below the editor is a 'PROBLEMS' and 'OUTPUT' panel. The 'OUTPUT' panel shows the following text:

```

*** Db2 Explained v0.0.21 STARTED ***
Analyzing SQL: select * from sysibm.systables where name like 'Ph...
Using AI Provider: Groq (qwen/qwen3-32b)
Step 1: Running EXPLAIN...
Step 1 Complete. Got 1 plan rows.
Step 2: Getting Recommendations from AI...
Step 2 Complete.
Report Generated.
  
```

The screenshot shows a Db2 Developer Explorer window with a workspace titled 'Untitled (Workspace)'. The main pane displays a 'Db2 Explain Report' generated on 2/11/2026 at 11:48:16 AM using the AI model 'qwen/qwen3-32b'. The SQL statement being analyzed is 'select * from sysibm.systables where name like 'Phill%''.

The 'Explain Plan' section contains the following table:

QBLOCKNO	PLANNO	TNAME	ACCESSTYPE	MATCHCOLS	INDEXONLY	ACCESSCREATOR	ACCESSNAME	COLUMN_FN_EVAL	CORRELATION
1	1	SYSTABLES	I	0	N	SYSIBM	DSNDTX01		

The 'AI Recommendations' section includes a warning icon and the following text:

⚠ Inefficient Index Usage for LIKE Predicate
The query uses index DSNDTX01 on SYSIBM.SYSTABLES, but MATCHCOLS=0 indicates no index columns are being used to filter rows. The LIKE 'Phill%' predicate on the NAME column is not effectively supported by the existing index structure. A dedicated index on (NAME) would improve performance by enabling direct access to matching rows.

*** Db2 Explained v0.0.21 STARTED ***

Analyzing SQL: select * from sysibm.systables where name like 'Ph...

Using AI Provider: Groq (qwen/qwen3-32b)

Step 1: Running EXPLAIN...

Step 1 Complete. Got 1 plan rows.

Step 2: Getting Recommendations from AI...

Step 2 Complete.

Report Generated.

Inefficient Index Usage for LIKE Predicate

The query uses index DSNCTX01 on SYSIBM.SYSTABLES, but MATCHCOLS=0 indicates no index columns are being used to filter rows. The LIKE 'Phill%' predicate on the NAME column is not effectively supported by the existing index structure. A dedicated index on (NAME) would improve performance by enabling direct access to matching rows.

Full Index Scan Instead of Index-Only Scan

The explain plan shows INDEXONLY=N, meaning the query requires additional table data not covered by the index. Creating a composite index on (NAME, all selected columns) could enable an index-only scan, reducing I/O.

Index Access Path Used

The optimizer selected an index access path (ACCESSTYPE=I) instead of a table scan, which is appropriate for the small table size (CARD=11348 rows). However, the current index is not optimized for the LIKE predicate.

*** Db2 Explained v0.0.21 STARTED ***

Analyzing SQL: select * from sysibm.systables where name like 'Ph...

Using AI Provider: Gemini (gemini-2.0-flash)

Step 1: Running EXPLAIN...

Step 1 Complete. Got 1 plan rows.

Step 2: Getting Recommendations from AI...

Step 2 Complete.

Report Generated.

⚠️ Index Scan with LIKE predicate on NAME

The query uses a LIKE predicate with a leading wildcard ('Phill%') on the NAME column. The existing index DSNDTX01 is being used, as indicated by the 'ACCESSTYPE': 'I' and 'ACCESSNAME': 'DSNDTX01'. However, because of the leading wildcard, DB2 cannot use the index effectively to directly locate the rows. It likely scans the index and then filters the rows. A more effective approach would be to create a new index on the NAME column. Consider creating an index specifically on the NAME column to optimize LIKE queries with leading fixed characters. This new index would minimize the number of index entries scanned.

ℹ️ Missing Index Recommendation for NAME Column

A new index on the NAME column of the SYSIBM.SYSTABLES table would significantly improve the performance of the query. This index should be defined as: ``CREATE INDEX IX_SYSTABLES_NAME ON SYSIBM.SYSTABLES (NAME);`` This index is especially beneficial due to the NAME column being used in the WHERE clause with a LIKE predicate. While the LIKE predicate has a trailing wildcard, DB2 can utilize this new index much better than the existing ones.

ℹ️ Check Index Usage After Creation

After creating the index IX_SYSTABLES_NAME, re-run the EXPLAIN PLAN for the query to ensure that DB2 is using the newly created index. If DB2 doesn't automatically pick up the new index, consider rebinding the application package or using a query hint to force the use of the index.

✅ Existing Index Usage

The explain plan shows the query is using an index (ACCESSTYPE = 'I'). While the current index is not optimal, the query isn't performing a full table scan, which is good. The addition of a more appropriate index will improve performance further.

*** Db2 Explained v0.0.22 STARTED ***

Analyzing SQL: select * from sysibm.systables where name like 'Ph...

Using AI Provider: OpenAI (gpt-3.5-turbo)

Step 1: Running EXPLAIN...

Step 1 Complete. Got 1 plan rows.

Step 2: Getting Recommendations from AI...

Step 2 Complete.

Report Generated.

AI Recommendations

Missing Index Recommendation

Based on the SQL query 'select * from sysibm.systables where name like 'Phill%', consider creating an index on the 'name' column of the 'SYSIBM.SYSTABLES' table to improve query performance. The index should be non-clustered and include the 'name' column. This will allow for faster retrieval of rows that match the 'name' condition.

*** Db2 Explained v0.0.22 STARTED ***

Analyzing SQL: select * from sysibm.systables where name like 'Ph...

Using AI Provider: OpenAI (gpt-4.1)

Step 1: Running EXPLAIN...

Step 1 Complete. Got 1 plan rows.

Step 2: Getting Recommendations from AI...

Step 2 Complete.

Report Generated.

AI Recommendations

⚠️ LIKE predicate with leading wildcard prevents index matching

The query uses 'name like 'Phill%'' which allows index matching only if an index starts with the NAME column. However, the Explain Plan shows MATCHCOLS=0, indicating no columns of the index are being used for matching. This means the index DSNDTX01 (likely on CREATOR, NAME) is not being used efficiently for this predicate.

⚠️ Index scan without column match leads to inefficiency

ACCESSTYPE='I' with MATCHCOLS=0 indicates a non-matching index scan (index screening), which is less efficient than an index match. The query will scan the entire index and then apply the LIKE predicate, resulting in unnecessary I/O, especially as the table grows.

ℹ️ Table size is moderate but can impact performance

SYSIBM.SYSTABLES has 11,348 rows and 832 pages. While not extremely large, a non-matching index scan or tablespace scan can still be costly, especially if this query is run frequently or as part of a larger workload.

✅ No unnecessary sorts detected

The Explain Plan shows no sorts for uniqueness, join, order by, or group by. This is optimal for this query.

ℹ️ Existing indexes do not support efficient LIKE predicate

The available indexes (DSNDTX01, DSNDTX02, DSNDTX03, DSNDTX05) do not have NAME as the first column, or are multi-column indexes where NAME is not leading. For a LIKE 'prefix%' predicate, an index on NAME as the first column is required for index matching.

⚠️ Missing index on NAME column for efficient LIKE predicate

To optimize this query, create a new index on SYSIBM.SYSTABLES(NAME). This will allow Db2 to use index matching for the LIKE 'Phill%' predicate, resulting in a much more efficient index scan. Recommended index: `CREATE INDEX SYSIBM.X_SYSTABLES_NAME ON SYSIBM.SYSTABLES(NAME);`

+ is this a good db2 z/os access path QBLOCKNO PLANNO TNAME X
 ACESSTYPE MATCHCOLS INDEXONLY ACCESSCREATOR
 ACCESSNAME COLUMN_FN_EVAL CORRELATION_NAME
 CREATOR CTEREF EXPANSION_REASON HINT_USED
 JOIN_DEGREE JOIN_PGROUP_ID JOIN_TYPE MERGC
 MERGE_JOIN_COLS MERGN METHOD MIXOPSEQ OPTHINT
 PAGE_RANGE PARALLELISM_MODE PARENT_PLANNO
 PARENT_QBLOCKNO PER_STMT_ID PREFETCH
 PRIMARY_ACESSTYPE PROGNAME QBLOCK_TYPE QUERYNO
 REMARKS ROUTINE_ID SCAN_DIRECTION SECTNOI
 SORTC_GROUPBY SORTC_JOIN SORTC_ORDERBY
 SORTC_PGROUP_ID SORTC_UNIQ SORTN_GROUPBY SORTN_JOIN
 SORTN_ORDERBY SORTN_PGROUP_ID SORTN_UNIQ STMTTOKEN
 TABLE_DCCSID TABLE_ENCODE TABLE_MCCSID TABLE_SCCSID
 TABLE_TYPE TABNO TSLOCKMODE VERSION WHEN_OPTIMIZE 1 1
 SYSTABLES I O N SYSIBM DSNCTX01 SYSIBM O N N O O O O O D
 SYSLH200 SELECT 1 O F O N N N N N N N N N 1200 U 1208 367 T 1 IS
 |

AI Mode

AI Overview

Based on the

PLAN_TABLE output provided, this access path is **generally not optimal, but potentially acceptable** depending on the size of the SYSIBM.SYSTABLES catalog table.

Here is the breakdown of the key columns and why it is not the best path:

Key Access Path Indicators

- **TNAME:** SYSIBM.SYSTABLES (The system catalog table containing metadata about tables).
- **ACCESSTYPE:** I (Index Scan).
- **MATCHCOLS:** 0 (Non-matching Index Scan).
- **ACCESSNAME:** DSNDTX01 (A primary index on SYSIBM.SYSTABLES).
- **INDEXONLY:** N (Data page access is required).

Analysis

1. **Non-Matching Index Scan (ACCESSTYPE='I', MATCHCOLS=0):** This means Db2 is scanning the entire DSNDTX01 index from beginning to end to find the desired row(s). It is better than a table space scan (Table Space Scan), but much less efficient than a matching index scan (where MATCHCOLS > 0).
2. **INDEXONLY='N':** Db2 is fetching the index key and then, for every index entry, traversing to the data page to fetch additional columns.
3. **Context (SYSTABLES):** SYSIBM.SYSTABLES is usually not large, so a non-matching index scan might have a low cost and provide acceptable performance. However, if this query is executed frequently (e.g., in a loop), it will incur unnecessary I/O overhead.

Recommendation

Why is it not matching? You are likely missing a highly selective predicate in your WHERE clause on the leading columns of the DSNDTX01 index (likely DBNAME, NAME, or CREATOR).

If performance is poor: Add a specific WHERE clause to your query, such as WHERE NAME = 'MYTABLE' AND CREATOR = 'MYSHEMA', to turn MATCHCOLS into a positive number (1 or higher).

Conclusion: It is a functional but non-optimal access path. It is safe, but not designed for high-performance retrieval.

⚠️ AI Analysis Failed

Could not get recommendations from AI Provider: Groq API failed with status 413: {"error":{"message":"Request too large for model `qwen/qwen3-32b` in organization `org_01kezbyy4nftrt94w2xfar4fea9` service tier `on_demand` on tokens per minute (TPM): Limit 6000, Requested 7071, plea. Please check your API key."}}

DSN_FILTER_TABLE

QUERYNO	QBLOCKNO	PREDNO	TYPE	FILTER_FACTOR	SEARCHARG	TEXT	JOIN	AFTER_JOIN	ADDED_PRED
1	1	1	AND	1.0000000000000000	Y	(CAST(CAST("HAUPXB0"."DSNWFQB(02)". "NAME" AS ...	N		
1	1	2	IN	1.0000000000000000	N	"T"."DBNAME" IN (SELECT "SYSIBM"."SYSDATABASE". "...	N		
1	1	3	LIKE	0.0003324467688799	Y	"T"."NAME" LIKE 'Phill%'	N		
1	1	4	EQUAL	0.0015797787345946	Y	CAST(CAST("T"."DBNAME" AS VARCHAR(24)) AS CHAR(...	Y		
1	1	5	EQUAL	0.0003720237873495	Y	CAST(CAST("T"."TSNAME" AS VARCHAR(24)) AS CHAR(8...	Y		
1	1	6	EQUAL	0.0004608293529600	Y	"T"."DBNAME"="HAUPXB0"."DSNWFQB(02)". "NAME"	Y		
1	1	8	EQUAL	0.0004608293529600	Y	CAST(CAST("HAUPXB0"."DSNWFQB(02)". "NAME" AS V...	N		T
1	1	9	COMPOUND	0.0000882122258190	Y	COMPOUND: 3,6			
1	1	10	COMPOUND	0.0002079002151731	Y	COMPOUND: 4,5			
1	1	11	COMPOUND	0.0004608293529600	Y	COMPOUND: 6,8			
1	1	12	COMPOUND	0.0000881466694409	Y	COMPOUND: 3,4,5			
1	2	7	EQUAL	0.0004608293529600	Y	"T"."DBNAME"="SYSIBM"."SYSDATABASE". "NAME"	N		C

```
+ this is from dns_predicate_table, what is happening here?
"QUERYNO" "QBLOCKNO" "PREDNO" "TYPE"
"LEFT_HAND_SIDE" "LEFT_HAND_PNO" "LHS_TABNO"
"LHS_QBNO" "RIGHT_HAND_SIDE" "RIGHT_HAND_PNO"
"RHS_TABNO" "RHS_QBNO" "FILTER_FACTOR"
"BOOLEAN_TERM" "SEARCHARG" "JOIN" "AFTER_JOIN"
"ADDED_PRED" "REDUNDANT_PRED" "DIRECT_ACCESS"
"KEYFIELD" "EXPLAIN_TIME" "CATEGORY" "CATEGORY_B"
"TEXT" "PRED_ENCODE" "PRED_CC SID" "PRED_MCCSID"
"MARKER" "PARENT_PNO" "NEGATION" "LITERALS" "CLAUSE"
"GROUP_MEMBER" "ORIGIN" "UNCERTAINTY" "SECTNOI"
"COLLID" "VERSION" "EXPANSION_REASON" "PER_STMT_ID"
"AP_PLANID" 1 1 1 "AND" "" 8 0 0 "" 0 0 0
1.00000000000000000000 "Y" "Y" "N" "" "" "N" "N" "N" "2026-02-
16-20.42.19.510000" -1 0 "( CAST(
CAST(""HAUPXBO"". ""DSNWFQB(02)"". ""NAME"" AS
VARCHAR(24)) AS C" "" 0 0 "N" 0 "N" ""Phill%"" "WHERE" "DJJ1"
"" 0.0000000000 0 "NULLID" "" "" 0
"X'00E240C4DECA71B30E000000008100001"" 1 1 2 "IN"
"DBNAME" 0 1 1 "SUBQUERY" 0 0 2 1.00000000000000000000 "Y"
```



AI Mode

Db2 Rewrote Your Query (Transitive Closure)

Look at **PREDNO 8**.

• **ADDED_PRED = 'T'**: This stands for **Transitive Closure**.

• **What happened**: You likely joined Table A to Table B, and Table B to Table C. Db2 realized: "If A=B and B=C, then A=C."

• **Result**: Db2 automatically generated PREDNO 8 adds a direct filter or join condition to a table that you didn't explicitly write. This allows Db2 to filter rows earlier (at Stage 1) or use an improved index.

2. Complex Index Probing (Compound Predicates)

Rows **9, 10, 11, and 12** have TYPE = COMPOUND.

• These are not standard WHERE clauses. They represent Db2 calculating the filtering power of **combinations** of columns.

• **Example (Row 12)**: COMPOUND: 3,4,5.

- This means Db2 is testing an index that covers the columns from predicates 3, 4, and 5 together (likely NAME, DBNAME, and TSNAME).
- It's verifying if using all three columns as an index key is cheaper than just using one.

3. The "Driving" Filter

Look at **PREDNO 3** (LIKE, NAME starts with 'Phill%').

• **FILTER_FACTOR = 0.00033**: This is extremely low (0.03%).

• **Impact**: Because this is so selective, Db2 will almost certainly use this as the starting point. It will check the index for names starting with "Phill" before doing any joins.

4. Subquery Transformation

• **PREDNO 2** (IN): This is your original IN (SELECT ...) subquery.

• **PREDNO 7** (ADDED_PRED = 'C'): 'C' stands for **Check Constraint** or **Correlated**. Db2 pushed a predicate *into* the subquery (QBLOCKNO 2) to correlate it with the outer table efficiently.

Summary

Db2 took your query, realized the NAME LIKE 'Phill%' filter was the strongest path, generated extra join conditions (ADDED_PRED = 'T') to skip redundant steps, and then evaluated several multi-column index combinations (COMPOUND) to ensure it picked the most efficient index.

Influence access paths

<https://www.ibm.com/docs/en/db2-for-zos/13.0.0?topic=paths-influencing-access-path-selection>

- **Preparing to influence access paths**

You can specify whether a Db2 subsystem applies optimization hints and other methods for influencing the selection of access paths for SQL statements.

- **Overriding predicate selectivities at the statement level**

You can override the default selectivity estimates that Db2 uses to select access paths for SQL statements.

- **Specifying optimization parameters at the statement level**

You can customize the values of certain optimization-related subsystem parameters and bind options for all instances of particular SQL statements within particular scopes by creating statement-level optimization parameters.

- **Specifying access paths at the statement level**

You can suggest that Db2 uses a certain access path for all instances of a particular SQL statement within a specified scope by creating statement-level access paths.

- **Working with input tables for the BIND QUERY command**

The BIND QUERY command uses values that you populate in certain input tables to populate catalog tables with data that influences access path selection for matching SQL statements.

- **Specifying access paths in a PLAN_TABLE instance**

You try to enforce a particular access path for a SQL statement that is issued by a specific single authorization ID by creating PLAN_TABLE access paths.

- **Validation of specified access paths**

Db2 cannot always use access paths that you specify. When a specified access path cannot be used, Db2 marks the specified access path as invalid.

- **Limitations on specified access paths**

Db2 cannot always apply the information that you provide to influence access path selection.

- **Interactions of methods for influencing access paths**

Statement level-methods for influencing access path selection and PLAN_TABLE optimization hints can be used in combination for the same statements. However, certain restrictions apply.

- **Modifying catalog statistics to influence access path selection**

If you have the proper authority, you can influence access path selection by using an SQL statements to change statistics values in the catalog. However, doing so is not generally recommended, except as a last resort.



```
RUNSTATS TABLESPACE DB1.TS1  
TABLE (TABLE_A) INDEX (ALL)
```

```
RUNSTATS TABLESPACE DB1.TS1  
INDEX (ALL)  
FREQVAL NUMCOLS 1 COUNT 10  
FREQVAL NUMCOLS 2 COUNT 10  
FREQVAL NUMCOLS 3 COUNT 10  
FREQVAL NUMCOLS 4 COUNT 10  
FREQVAL NUMCOLS 5 COUNT 10
```

Disable list pre-fetch

Select ...

From ...

Where ...

Fetch first 1 rows only

Get AI to recommend and create indexes....



IDUG

2026 Australia **Db2** Tech Conference