LDUG 4th March 2026

# Explaining EXPLAIN

Gareth Copplestone-Jones
*gcj@triton.co.uk*

# EXPLAIN – The Very Basics

- What does EXPLAIN do?
  - EXPLAIN externalises access path information to a set of user-defined EXPLAIN tables
- What is an access path?
  - An access path (or access plan) is how Db2 accesses the data requested by the application via an SQL query
- How do I create the EXPLAIN tables?
  - Use the SQL in SDSNSAMP member DSNTESC to create the EXPLAIN tables using your chosen qualifier
  - Or use the ADMIN_EXPLAIN_MAINT stored procedure to create them
- How do I obtain EXPLAIN output?
  - Via the EXPLAIN SQL statement – see later slides
  - Via the EXPLAIN options on BIND/REBIND – see later slides


*** Triton Consulting and Isbank - Commercial and In Confidence ***

# Access Plan-related EXPLAIN Tables

PLAN_TABLE

Access plan for the SQL statement

Can be used for optimization hints

DSN_STATEMNT_TABLE

Cost estimates in service units and milliseconds

DSN_FUNCTION_TABLE

How functions are resolved

DSN_DETCOST_TABLE

Detailed cost estimates for the mini-plans in a query

DSN_FILTER_TABLE

How predicates are used

DSN_PGRANGE_TABLE

Information about qualified partitions for all page range scans in a query.

DSN_PGROUP_TABLE

Information about the parallel groups in a query

DSN_PREDICAT_TABLE

Information about all predicates in a query

DSN_PTASK_TABLE

Information about all parallel tasks in a query

DSN_QUERY_TABLE

SQL statement text before and after query transformation

DSN_SORTKEY_TABLE

Information about sort keys for all sorts required by a query

DSN_SORT_TABLE

Information about the sort operations required by a query

DSN_STRUCT_TABLE

Information about all query blocks in a query

DSN_VIEWREF_TABLE

Information about all views and MQTs used to process a query

DSN_KEYGTDIST_TABLE

Non-uniform index expression statistics obtained dynamically by the Optimizer

DSN_COLDIST_TABLE

Non-uniform column group statistics obtained dynamically from non-index leaf pages

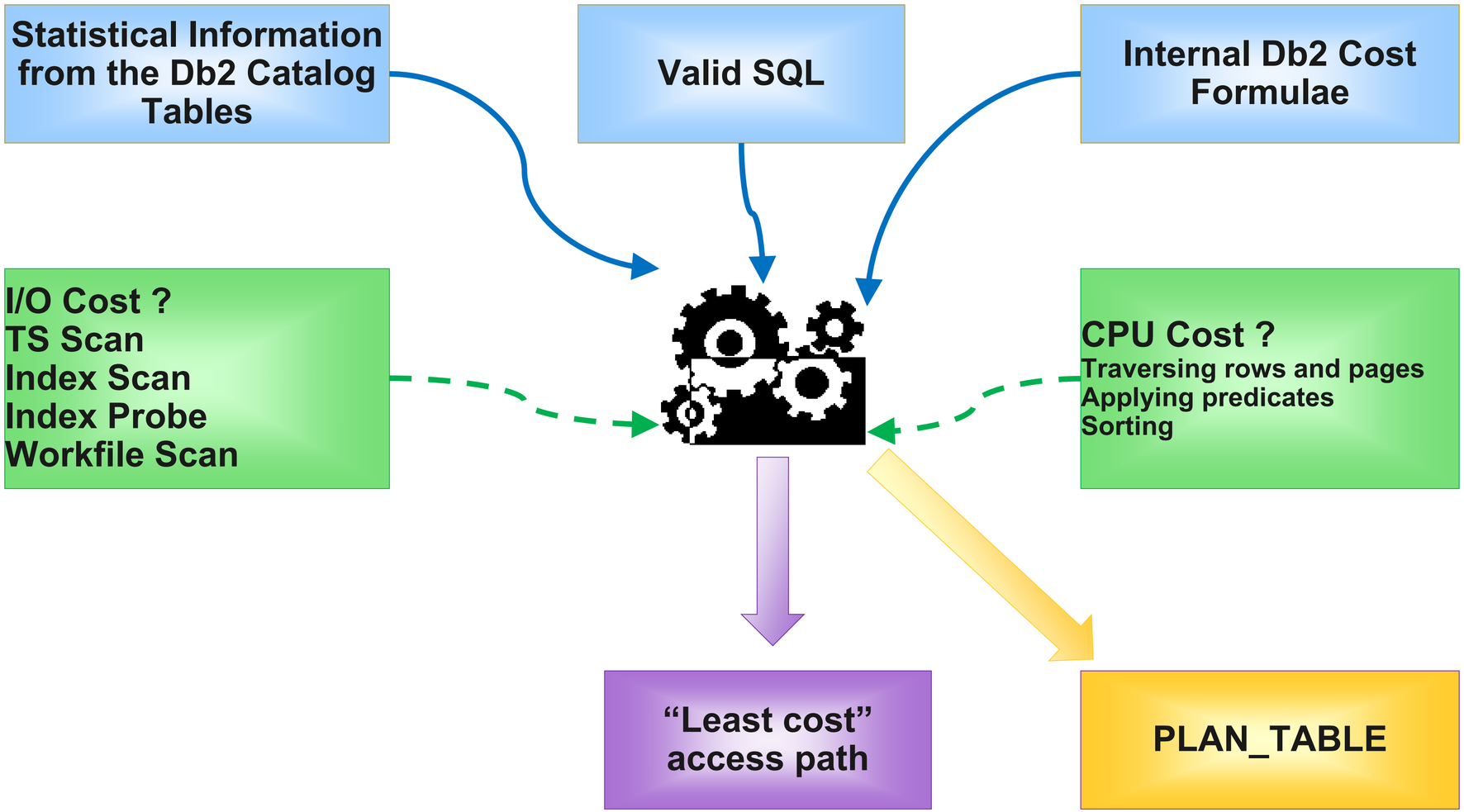
DSN_QUERYINFO_TABLE

Information about automatic query rewrite, considered MQTs, and query block acceleration

DSN_PREDICATE_SELECTIVITY

Information about the selectivity of predicates that are used for access path selection

# The Db2 Optimizer



# Obtaining Explain Information – BIND/REBIND and PREPARE

- ‘Standard’ BIND/REBIND
  - EXPLAIN(YES) produces a new package and writes access path information to the EXPLAIN tables
  - EXPLAIN(ONLY) does not produce a new package but writes access path information to the EXPLAIN tables
    - If your application contains dynamic SQL, EXPLAIN information will NOT be produced when the statement is prepared
- PREPARE and the CURRENT EXPLAIN MODE special register
  - SET CURRENT EXPLAIN MODE = NO – the default
  - SET CURRENT EXPLAIN MODE = YES dynamic SQL statements run, and access plan information for explainable SQL statements is captured and written to the EXPLAIN tables
  - SET CURRENT EXPLAIN MODE = EXPLAIN dynamic SQL statements do not execute, but access plan information is captured and written to the EXPLAIN tables
  - Access path information is captured at every re-optimization caused by the REOPT ONCE, ALWAYS and AUTO options

# Obtaining Explain Information – The SQL EXPLAIN Statement

- EXPLAIN PLAN SET QUERYNO=*nnn* FOR *<sql_statement>*
  - Asks Db2 to explain a single SQL statement by supplying the SQL statement text
  - User-supplied QUERYNO integer value can be used to identify the statement
- EXPLAIN STMTCACHE STMTID *value*
  - Asks Db2 to explain a single SQL statement cached in the Dynamic Statement Cache
  - STMTID obtained by EXPLAIN STMTCACHE ALL and matching SQL text
- EXPLAIN STMTCACHE STMTTOKEN *value*
  - Asks Db2 to explain a single SQL statement cached in the Dynamic Statement Cache
  - STMTTOKEN supplied by the application program itself and is recorded in DSN_STATEMENT_CACHE_TABLE
- EXPLAIN PACKAGE COLLECTION *ccc* PACKAGE *ppp* VERSION *vvv*
  - Asks Db2 to extract access path information from the Db2 Catalog/Directory for an existing package
  - Introduced in Db2 10 and valid for packages bound in Db2 9 or later
- EXPLAIN STMTCACHE ALL
  - Causes Db2 to write performance metrics gathered for statements in the DSC to DSN_STATEMENT_CACHE_TABLE
  - Does not generate information about access paths

# Retrieving Access Plan Information from PLAN_TABLE

- Access plan information can be obtained after EXPLAIN processing by querying the EXPLAIN tables
- The most commonly used one of these is PLAN_TABLE. which should be sorted by:
  - COLLID – the collection name
  - PROGNAME – the package name
  - VERSION – the package version
  - EXPLAIN_TIME – when EXPLAIN information was captured
    - For cached dynamic SQL statements. when the statement entered the cache,
    - For static SQL statements, when the statement was bound
    - For non-cached dynamic SQL statements, when EXPLAIN was executed
  - QUERYNO – identifies the statement being explained
    - The QUERYNO clause on EXPLAIN
    - The QUERYNO clause on a DML statement
    - The source application program line number – determined by the Db2 pre-compiler
  - QBLOCKNO – the query block position in statement
    - One query block per sub-select
    - ‘1’ is the outer-most query block
  - PLANNO - the execution order of the steps in a query block
  - MIXOPSEQ – the order for multiple index operations

# Joining EXPLAIN Tables

- All access-plan related EXPLAIN Tables include the following columns, which can be used to join them:
  - QUERYNO – identifies the statement being explained (see previous slide)
  - APPLNAME – the plan name for embedded SQL statements
  - COLLID – the collection name
  - PROGNAME – the package name
  - VERSION – the package version
  - EXPLAIN_TIME – when EXPLAIN information was captured
  - SECTNOI – the statement section number, derived from the SECTNOI column in the SYSPACKSTMT and SYSSTMT catalog tables

- Note – you must maintain the EXPLAIN tables yourself, by selectively deleting old rows that are no longer needed

# Reading PLAN_TABLE and DSN_STATEMNT_TABLE

```
SELECT SUBSTR(PL.COLLID,1,10) AS COLLID,
     SUBSTR(PL.PROGNAME,1,10) AS PROGNAME,
     DATE(PL.EXPLAIN_TIME) AS DATE,     TIME(PL.EXPLAIN_TIME) AS TIME,
     COUNT(PL.QUERYNO) AS "ST COUNT",
     DEC(SUM(ST.TOTAL_COST),8,2) AS "TOT COST"
FROM SJD.PLAN_TABLE PL, SJD.DSN_STATEMNT_TABLE ST
WHERE PL.PROGNAME = ST.PROGNAME AND PL.COLLID = ST.COLLID
AND PL.EXPLAIN_TIME = ST.EXPLAIN_TIME AND PL.QUERYNO = ST.QUERYNO
GROUP BY PL.COLLID, PL.PROGNAME, PL.EXPLAIN_TIME
ORDER BY PL.PROGNAME;
```

| COLLID | PROGNAME | DATE | TIME | ST COUNT | TOT COST |
|---|---|---|---|---|---|
| GLWGZJ | ACTSEL | 2016-11-23 | 10.22.27 | 6 | 0.10 |
| GLWGZJ | DPTADD | 2016-11-23 | 10.22.28 | 25 | 0.00 |
| GLWGZJ | DPTBAL | 2016-11-23 | 10.22.28 | 43 | 11.52 |
| GLWGZJ | DPTDEL | 2016-11-23 | 10.22.28 | 113 | 48.38 |
| GLWGZJ | DPTLCK | 2016-11-23 | 10.22.28 | 17 | 41.50 |
| GLWGZJ | DPTMGR | 2016-11-23 | 10.22.28 | 19 | 0.16 |
| GLWGZJ | DPTSEL | 2016-11-23 | 10.22.27 | 41 | 710.71 |
| GLWGZJ | DPTUPD | 2016-11-23 | 10.22.28 | 19 | 0.13 |
| GLWGZJ | DPTUPR | 2016-11-23 | 10.22.28 | 19 | 20.23 |

Estimated total cost in elapsed time (CPU + I/O)

# PLAN_TABLE Columns – Some Basics

- ACCESSTYPE – the technique used to access the table
  - Too many values to list – see the following slides
- METHOD
  - Join and sort indicators
    - 0: First table accessed
    - 1: Nested loop join
    - 2: Merge scan join (sort merge join)
    - 3: Sort
    - 4: Hybrid join
- MATCHCOLS
  - For index access, the number of (leading) index columns used for predicate matching
- PREFETCH – whether whether data pages are read in advance by prefetch:
  - D Optimizer expects dynamic prefetch
  - S Pure sequential prefetch
  - L Prefetch through a page list
  - U List prefetch with an unsorted RID list
  - blank Unknown or no prefetch

# More About Prefetch

- Dynamic prefetch (PREFETCH = ‘D’)
  - Db2 uses a sequential detection algorithm to determine whether data pages accessed via an index are being read sequentially
  - Db2 also uses the algorithm to determine with index leaf pages are being read in key-sequential order
    - More likely for an organized index
    - Synchronous I/O still occurs for the non-lead pages
- Sequential prefetch (PREFETCH = ‘S’)
  - Exclusively used for table space scans
- List prefetch (PREFETCH = ‘L’ or ‘U’)
  - L – prefetch using a sorted list of record identifiers (RIDs)
  - U – prefetch using an unsorted RID list

# Single Table Access Paths – Tablespace Scan

- Tablespace scan – Db2 reads the entire tablespace to return rows to the application – for example, because:
  - Using an index is not possible because no index is available, or because none of the predicates match the columns in an available index
  - A large proportion of the rows in the table qualify, therefore using an index is not efficient
  - ACCESSTYPE = ‘R’ (relational scan)

```
SELECT DEPT_NO, DEPT_NAME, MGR_NO, ADMR_DEPT, LOCATION, EMPLOYEE_CNT
FROM GLWTDPT
```

```
+-------------------------------------------------------------+
| QNO | QBN | PNO | MTH | CREATOR  |  TABLE   | ATYP | LM | PF |
+-------------------------------------------------------------+
| 600 |  1  |   1 |   0 | GLWGZJ   | GLWTDPT  | R    | IS | S  |
+-------------------------------------------------------------+
```

# Single Table Access Paths – Index Scans

- Matching index scan – Db2 uses an index to retrieve the rows requested by the application:
  - ACCESSTYPE = ‘I’ (index access)
  - MATCHCOLS > 0 (predicates are specified on one or more leading columns of the index)

```
SELECT LASTNAME, FIRSTNME, EMP_NO
FROM GLWGZJ.GLWTEMP
WHERE LASTNAME = ? AND FIRSTNME = ? AND EMP_NO > ?;
```

```
+-------------------------------------------------------------------------------
| QNO | QBN | PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  | MC | IXO |
+-------------------------------------------------------------------------------+
| 777 |   1 |   1 |   0 | GLWGZJ  | GLWTEMP | I    | GLWGZJ    | GLWXEMP1 |  3 | Y   |
+-------------------------------------------------------------------------------+
```

- Non-matching index scan – all the index keys and their RIDs are read because the leading index column doesn’t provide filtering
  - ACCESSTYPE = ‘I’ (index access)
  - MATCHCOLS = 0

# Single Table Access Paths – One-Fetch Index Access

- One-fetch index access – Db2 uses an index to retrieve a single row where the query includes MIN or MAX:
  - ACCESSTYPE = ‘I1’
  - MATCHCOLS = 0
  - No GROUP BY
  - Aggregate function must be on a column in the index

```
SELECT MAX(EMP_NO)
FROM GLWGZJ.GLWTEMP;
```

```
+-------------------------------------------------------------------------------------------+
| QNO | QBN | PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  | MC | IXO |
+-------------------------------------------------------------------------------------------+
| 777 |   1 |   1 |   0 | GLWGZJ  | GLWTEMP | I1   | GLWGZJ    | GLWXEMP3 |  0 | Y   |
+-------------------------------------------------------------------------------------------+
```

# Single Table Access Paths – IN-list Index Scan

- IN-list index scan – special case of matching index scan:
  - A single indexable IN predicate is used as a matching equal predicate
  - A series of matching index scans with the IN predicate values being used for each matching index scan
  - ACCESSTYPE = ‘N’
  - MATCHCOLS > 0

```
SELECT * FROM GLWGZJ.GLWTEPA
WHERE PROJ_NO = ? AND ACT_NO IN (?, ?, ?)
AND EMP_NO = ? ;
```

```
+-------------------------------------------------------------------------------------------+
| QNO | QBN | PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  | MC | IXO |
+-------------------------------------------------------------------------------------------+
| 777 |   1 |   1 |   0 | GLWGZJ  | GLWTEPA | N    | GLWGZJ    | GLWXEPA1 |  3 | N   |
+-------------------------------------------------------------------------------------------+
```

```
WHERE (PROJ_NO = ? AND ACT_NO = ? AND EMP_NO = ?)
   OR (PROJ_NO = ? AND ACT_NO = ? AND EMP_NO = ?)
   OR (PROJ_NO = ? AND ACT_NO = ? AND EMP_NO = ?)
```

# Single Table Access Paths – IN-list Direct Table Access

- IIN-list direct access – in-memory tables used to process one or more IN-list predicates as matching predicates:
  - PLAN_TABLE contains one row for each predicate processed through in-memory tables
  - ACCESSTYPE = 'IN' for the in-memory tables and 'I' for the base table
  - MATCHCOLS > 0
  - The tables are joined by nested-loop join – see later discussion on table joins

```
SELECT * FROM GLWGZJ.GLWTEMP WHERE WORKDEPT IN (?, ?, ?) AND JOB IN (?, ?, ?);

+-------------------------------------------------------------------------------+
|PNO | MTH | CREATOR |  TABLE       | ATYP | IXCREATOR |  IXNAME   | MC | IXO | PF | TTYP |
+-------------------------------------------------------------------------------+
|  1 |   0 | JONESGT | DSNIN002(01) | IN   |           |           |  0 | N   |    | I    |
|  2 |   1 | GLWGZJ  | GLWTEMP      | I    | GLWGZJ    | GLWXEMP2  |  1 | N   | L  | T    |
+-------------------------------------------------------------------------------+
```

  - PREFETCH = 'L' – Db2 uses list prefetch to identify data pages to read by a obtaining a list of record identifiers (RIDs) from the matching index entries
  - Column WORKDEPT is indexed by GLWXEMP2

# Single Table Access Paths – Multiple Index Access

- Db2 uses more than one index to access a table, when no single index provides efficient access or when a combination of indexes provides efficient access

```
SELECT * FROM GLWGZJ.GLWTEMP WHERE WORKDEPT = ? AND JOB = ?;
```

| MTH | CREATOR | TABLE | ATYP | IXCREATOR | IXNAME | MC | IXO | PF | MXS |
|---|---|---|---|---|---|---|---|---|---|
| 0 | GLWGZJ | GLWTEMP | M | | | 0 | N | L | 0 |
| 0 | GLWGZJ | GLWTEMP | MX | GLWGZJ | GLWXEMP2 | 0 | Y | | 1 |
| 0 | GLWGZJ | GLWTEMP | MX | GLWGZJ | GLWXEMP5 | 0 | Y | | 2 |
| 0 | GLWGZJ | GLWTEMP | MI | | | 0 | N | | 3 |

  - ACCESSTYPE = ‘M’: multiple index scan, followed by other rows
  - ACCESSTYPE = ‘MX’: index scan on the named index
  - ACCESSTYPE = ‘MI’: intersection of multiple indexes (‘AND’)
  - ACCESSTYPE = ‘MU’: union of multiple indexes (‘OR’)
  - A RID list is constructed for each index; the RID intersections produce a list of qualified RIDs used to retrieve the rows using list prefetch

# JOIN Processing

- A join reads rows from more than one table and combines then in the form of *composite tables* and *new tables*:
  - A *composite table* represents the result of accessing the first or *outer* table in join processing
  - A *new table* is the joined-to or *inner* table in join processing
- A table may be joined to itself so may be both the composite (outer) and new (inner) table
- A join matches a row of one table with a row of another table using a join condition
  - An *inner join* discards rows of either table that do not match any row of the other table
  - An *outer join* keeps unmatched rows of one or the other table, or of both
- Join types:
  - Nested loop join (METHOD = 1)
  - Merge scan join (METHOD = 2), also know as sort merge join
  - Hybrid Join (METHOD = 4)
  - Star schema access (JOIN_TYPE = 'S' or 'P'), also known as star join – not covered in this presentation

# Nested Loop Join – METHOD = 1

- In a nested loop join, Db2:
  - Scans the *composite* (outer) table.
  - For each qualifying row in that table (matching the *local* predicates), Db2 searches for matching rows of the *new* (inner) table
- Nested loop join is more likely to be used when one or more of the following is true:
  - The outer table is small
  - Predicates that filter efficiently limit the number of outer table qualifying rows
  - There is an efficient, highly clustered index on the join columns of the inner table
  - The number of data pages accessed in the inner table is small
  - No join columns exist
- Nested loop join can use a *sparse index* on the inner table:
  - Db2 dynamically creates a sparse index to search the work file built on the inner table
  - This makes access to the inner table more efficient when there is no efficient index on the join column
  - Db2 can avoid sorting the composite table when it is relatively large
  - Indicated in PLAN_TABLE with PRIMARY_ACCESSTYPE = ‘T’

# Nested Loop Join Example

```
SELECT EMP.FIRSTNME, EMP.LASTNAME, EMP.WORKDEPT,
DEPT.DEPT_NAME
FROM GLWGZJ.GLWTEMP EMP, GLWGZJ.GLWTDPT DEPT
WHERE EMP.WORKDEPT = DEPT.DEPT_NO
ORDER BY EMP.LASTNAME ;
```

```
+-------------------------------------------------------------+
| PNO | MTH | CREATOR |  TABLE   | ATYP | IXCREATOR |  IXNAME  |
+-------------------------------------------------------------+
|   1 |   0 | GLWGZJ  | GLWTEMP  | I    | GLWGZJ    | GLWXEMP1 |
|   2 |   1 | GLWGZJ  | GLWTDPT  | R    |           |          |
+-------------------------------------------------------------+
+-----------------------------------------------+
| PF | SNU | SNJ | SNO | SNG | SCU | SCJ | SCO | SCG |
+-----------------------------------------------+
|    | N   | N   | N   | N   | N   | N   | N   | N   |
| S  | N   | Y   | N   | N   | N   | N   | N   | N   |
+-----------------------------------------------+
```

# Merge Scan Join – METHOD = 2

- There must be join predicates on the two tables
  - E.g. TABLE1.COL1 = TABLE2.COL2
  - The two columns must have the same data type and length
- In a merge scan join, Db2 scans both tables in the order of the join columns
  - If there are no efficient indexes on the join columns to provide the order, Db2 might sort one or both of the tables
  - The inner table is always put into a work file
  - The outer table is put into a work file only if it must be sorted
  - Once in join column order, the tables are scanned and merged together
- Merge scan join is more likely to be used when one or more of the following is true :
  - The number of qualifying rows in the inner and outer table is large, and the join predicates
  - do not provide much filtering – that is, this is a many-to-many join.
  - The tables are large and have no indexes with matching columns
  - Few of the inner table columns are selected, allowing a Db2 sort to be used – the fewer the number of columns to be sorted, the more efficient the sort

# Merge Scan Join Example

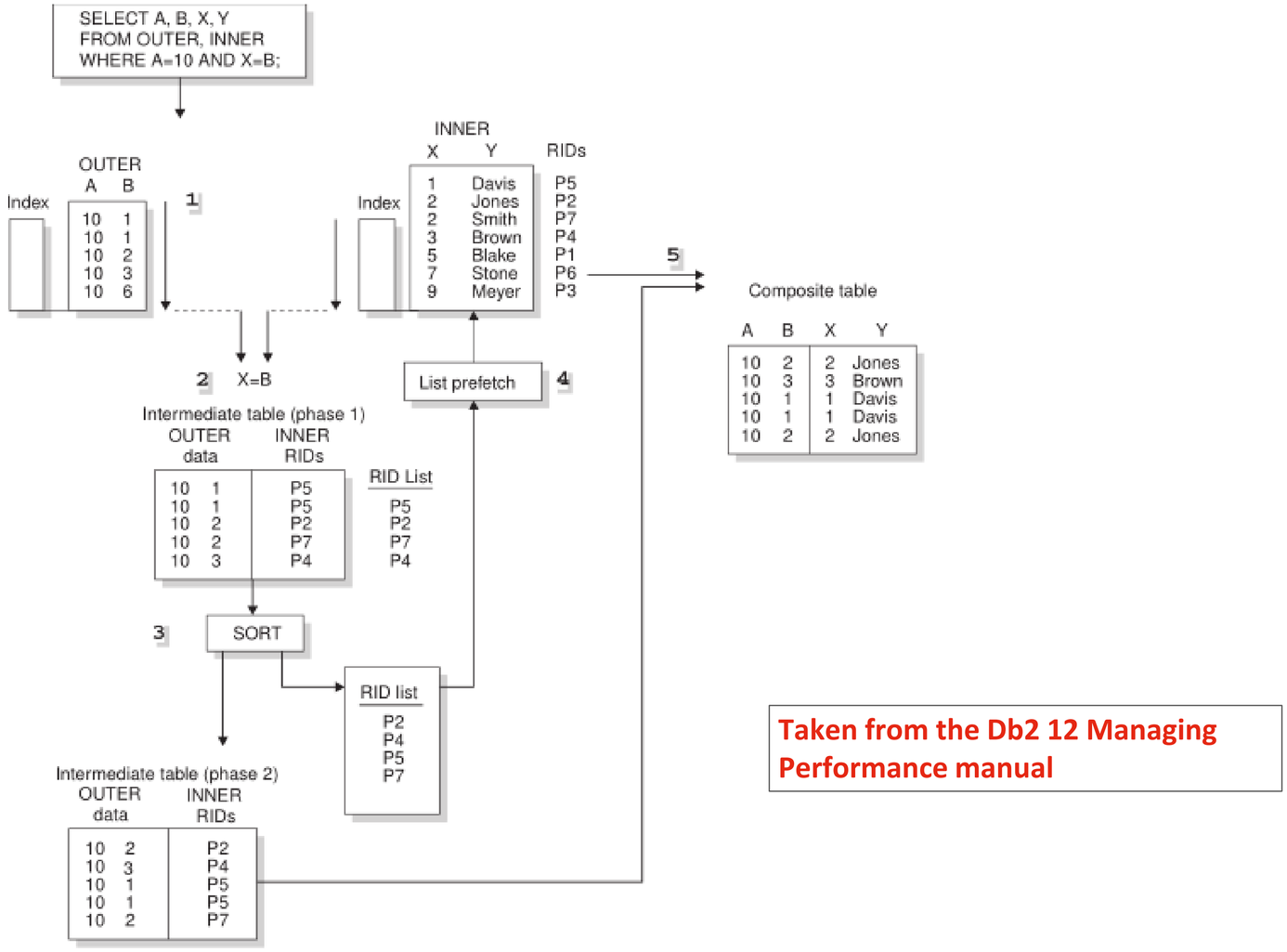
```
SELECT PJA.PROJ_NO, PJA.ACT_NO, PJA.ACTDESC,
EPA.EMP_NO
FROM GLWGZJ.GLWTPJA PJA, GLWGZJ.GLWTEPA EPA
WHERE PJA.ACT_NO = EPA.ACT_NO
AND EPA.EMENDATE < PJA.ACENDATE
ORDER BY PJA.PROJ_NO ;
```

| PNO | MTH | CREATOR | TABLE | ATYP | PF | SNU | SNJ | SNO | SNG | SCU | SCJ | SCO | SCG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | GLWGZJ | GLWTPJA | R | S | N | N | N | N | N | Y | N | N |
| 2 | 2 | GLWGZJ | GLWTEPA | R | S | N | Y | N | N | N | N | N | N |
| 3 | 3 | | | | | N | N | N | N | N | N | Y | N |

# Hybrid Join – METHOD = 4

- Hybrid join requires an index on the join column(s) of the inner table
- Hybrid join:
  - Scans the outer table
  - Reads the index on the inner table to find matching rows, extracts the RIDs and joins them with the outer table
    - The index of the inner table is scanned for every row of the outer table
  - Typically sorts the joined outer table and RIDs, creating a sorted RID list
    - If the index on the inner table is well-clustered, Db2 can skip this sort
  - Retrieves the data from the inner table, using list prefetch
  - Concatenates the data from the inner table and sorted outer table to create the final composite table
- Hybrid join can be used when:
  - A non-clustered index (or indexes) are used on the join columns of the inner table
  - The outer table has duplicate qualifying rows

# Hybrid Join – METHOD = 4 …




**Taken from the Db2 12 Managing Performance manual**

# Hybrid Join Example

```
SELECT EMP.WORKDEPT, EPA.PROJ_NO
FROM GLWGZJ.GLWTEMP EMP, GLWGZJ.GLWTEPA EPA
WHERE EMP.EMP_NO = EPA.EMP_NO
AND EMP.HIREDATE > ? ;
```

```
+-------------------------------------------------------------------+
| PNO | MTH | CREATOR |  TABLE   | ATYP | IXCREATOR |  IXNAME   | MC |
+-------------------------------------------------------------------+
|   1 |   0 | GLWGZJ  | GLWTEMP | R    |           |           |  0 |
|   2 |   4 | GLWGZJ  | GLWTEPA | I    | GLWGZJ    | GLWXEPA3 |  1 |
+-------------------------------------------------------------------+
+-----------------------------------------------+
| PF | SNU | SNJ | SNO | SNG | SCU | SCJ | SCO | SCG |
+-----------------------------------------------+
| S  | N   | N   | N   | N   | N   | N   | N   | N   |
| L  | N   | Y   | N   | N   | N   | Y   | N   | N   |
+-----------------------------------------------+
```

# Three-way Join Example

```
SELECT EPA.PROJ_NO, EPA.ACT_NO, PJA.ACTDESC, EMP.WORKDEPT
FROM GLWGZJ.GLWTEMP EMP, GLWGZJ.GLWTEPA EPA, GLWGZJ.GLWTPJA PJA
WHERE EMP.EMP_NO = EPA.EMP_NO
AND PJA.ACT_NO = EPA.ACT_NO
AND EMP.HIREDATE > ?
ORDER BY EMP.WORKDEPT ;
```

| PNO | MTH | CREATOR | TABLE | ATYP | IXCREATOR | IXNAME | MC |
|---|---|---|---|---|---|---|---|
| 1 | 0 | GLWGZJ | GLWTEMP | I | GLWGZJ | GLWXEMP2 | 0 |
| 2 | 1 | GLWGZJ | GLWTEPA | I | GLWGZJ | GLWXEPA3 | 1 |
| 3 | 1 | GLWGZJ | GLWTPJA | R | | | 0 |

| PF | SNU | SNJ | SNO | SNG | SCU | SCJ | SCO | SCG |
|---|---|---|---|---|---|---|---|---|
| | N | N | N | N | N | N | N | N |
| | N | N | N | N | N | N | N | N |
| | N | Y | N | N | N | N | N | N |

# Outer Joins

- An outer join keeps unmatched rows for one or both of the tables
- Column values from an unmatched row are represented with null
  - Left outer join keeps rows from the composite (outer) table
  - Right outer join keeps rows from the new (inner) table
  - Full outer join keeps rows from both tables
  - The following will return rows for departments without employees:

```
SELECT DEPT.DEPT_NO, DEPT.DEPT_NAME, EMP.EMP_NO,
EMP.FIRSTNME, EMP.LASTNAME
FROM GLWGZJ.GLWTDPT DEPT
LEFT OUTER JOIN GLWGZJ.GLWTEMP EMP
ON EMP.WORKDEPT = DEPT.DEPT_NO
ORDER BY DEPT.DEPT_NO;
```

```
+-------------------------------------------------------------------+
| PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  | MC | JT |
+-------------------------------------------------------------------+
|   1 |   0 | GLWGZJ  | GLWTDPT | I    | GLWGZJ    | GLWXDPT1 |  0 |    |
|   2 |   1 | GLWGZJ  | GLWTEMP | R    |           |          |  0 |  L |
+-------------------------------------------------------------------+
```

# EXPLAIN, Subqueries, and Materialisation of Views and Nested Table Expressions

- PLAN_TABLE shows the position and order in which queries are executed as query blocks (QBLOCKNO)
  - Subqueries may be re-written by Db2 as joins
    - The join access path is recorded in PLAN_TABLE
    - The query before and after transformation is recorded in DSN_QUERY_TABLE as XML data in a CLOB column
    - When a subquery is transformed into a join, the PLAN_TABLE rows for the transformed subquery have the same QBLOCKNO as the outer query
- When views or nested table expressions are materialised, their rows are put into a work file for processing like a table
  - TNAME contains the name of the view or nested table expression
  - TABLE_TYPE contains ‘W’

# Non-correlated Subquery

```
SELECT EMP.EMP_NO, EMP.FIRSTNME, EMP.LASTNAME
FROM GLWGZJ.GLWTEMP EMP
WHERE EMP.WORKDEPT IN
(SELECT DEPT.DEPT_NO
FROM GLWGZJ.GLWTDPT DEPT
WHERE DEPT.DEPT_NAME IN (?, ?, ?, ?, ?, ?)
AND DEPT.DEPT_NO IN
(SELECT PRJ.DEPT_NO FROM GLWGZJ.GLWTPRJ PRJ
WHERE PRJ.PROJ_NO IN (?, ?, ?, ?, ?, ?)
)) ;
```

| QBN | PNO | MTH | CREATOR | TABLE | ATYP | IXCREATOR | IXNAME | TT | QBT | PQB |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | JONESGT | DSNWFQB(03) | R | | | W | SELECT | 0 |
| 1 | 2 | 1 | GLWGZJ | GLWTDPT | I | GLWGZJ | GLWXDPT1 | T | SELECT | 0 |
| 1 | 3 | 1 | GLWGZJ | GLWTEMP | I | GLWGZJ | GLWXEMP2 | T | SELECT | 0 |
| 3 | 1 | 0 | GLWGZJ | GLWTPRJ | N | GLWGZJ | GLWXPRJ1 | T | NCOSUB | 1 |
| 3 | 2 | 3 | | | | | | ? | NCOSUB | 1 |

# Correlated Subquery

```
SELECT EMP.EMP_NO, EMP.FIRSTNME, EMP.LASTNAME
FROM GLWGZJ.GLWTEMP EMP
WHERE EXISTS
(SELECT 1 FROM GLWGZJ.GLWTDPT DEPT
WHERE DEPT.DEPT_NAME IN (?, ?, ?, ?, ?, ?)
AND DEPT.CREATED_BY = EMP.CREATED_BY
AND DEPT.DEPT_NO IN
(SELECT PRJ.DEPT_NO FROM GLWGZJ.GLWTPRJ PRJ
WHERE PRJ.PROJ_NO IN (?, ?, ?, ?, ?, ?)
AND PRJ.CREATED_TS = DEPT.CREATED_TS
AND PRJ.CREATED_TS = EMP.CREATED_TS
)) ;
```

| QBN | PNO | MTH | CREATOR | TABLE | ATYP | IXCREATOR | IXNAME | TT | QBT | PQB |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | GLWGZJ | GLWTEMP | R | | | T | SELECT | 0 |
| 2 | 1 | 0 | GLWGZJ | GLWTDPT | R | | | T | CORSUB | 1 |
| 3 | 1 | 0 | GLWGZJ | GLWTPRJ | N | GLWGZJ | GLWXPRJ1 | T | CORSUB | 2 |

# Wrapping Up Explain – UNION Type Operators

- QBLOCK_TYPE values
  - UNION = UNION operator
  - UNIONA = UNION ALL operator
  - INTERS = INTERSECT operator
  - INTERSA = INTERSECT ALL operator
  - EXCEPT = EXCEPT operator
  - EXCEPTA = EXCEPT ALL operator
  - The PLAN_TABLE row with one of these values will have a different QBLOCKNO to the other subselects, a null in TABLE_TYPE and blank in TNAME

# Explaining EXPLAIN

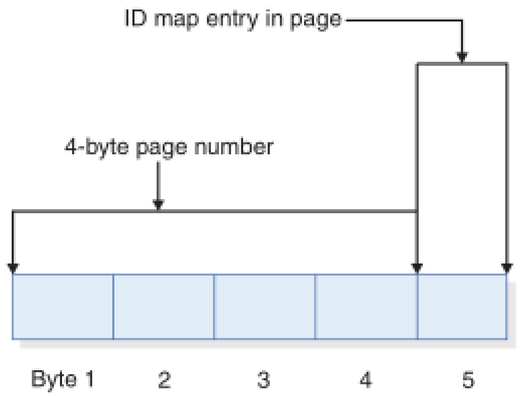
Gareth Copplestone-Jones

LDUG 4th March 2026

# List Prefetch

- Optimizer-selected list prefetch:
  - Db2 reads index entries, obtains a list of record identifiers (RIDs), sorts the RIDs into ascending page order and uses that list to prefetch data pages into the buffer pool
  - List prefetch does not use sequential detection
  - Db2 uses the RID pool to process the RID list for list prefetch
  - A RID is composed of four (segmented, classic partitioned, simple table spaces), five (LARGE and UTS table spaces) or seven (UTS PBR RPN) bytes
  - ID map entries:
    - Db2 stores rows in data pages
    - Each page has a page header and a page trailer
    - The page trailer contains one or more ID map entries – one for each row – that point to the offset within the data page for the row

RID size increases to 7 for Db2 UTS PBR tablespaces using relative page numbering



# More On List Prefetch

- List prefetch is typically used under the following conditions:
  - For an index with a cluster ratio lower than 80%
  - Sometimes for indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small for efficient sequential prefetch, but still requires multiple pages
  - Always for multiple index access
  - Always for a hybrid join
  - For updatable cursors when index columns might be updated.
  - For IN-list access using an in-memory table
- There are advantages and disadvantages of list prefetch – see the Db2 ‘Managing Performance’ manual for more information

# Db2 Data Row Sort Processing

- As well as RID sorting, Db2 sorts data rows – the result of the sort might, or might not, be materialized into a work file
- Two types of tables are sorted:
  - A *composite table* represents the result of accessing one or more tables in a query
    - For a single table, there is one composite table
    - If one or more joins are involved, an *outer composite table* consists of the intermediate result rows from the previous join step.
  - A *new table* (or *inner table*) can be sorted in a join operation and is the joined-to table

## PLAN_TABLE columns:

**For composite tables**

**SORTC_UNIQ** – sort to remove duplicates
**SORTC_JOIN** – sort for merge scan join or hybrid join
**SORTC_ORDERBY**
**SORTC_GROUPBY**

**For new tables**

**SORTN_UNIQ** – sort to remove duplicates
**SORTN_JOIN** – sort for merge scan join or hybrid join
**SORTN_ORDERBY**
**SORTN_GROUPBY**

# More On Db2 Data Row Sort Processing

- Sorts for GROUP BY and ORDER BY are indicated by SORTC_ORDERBY and SORTC_GROUPBY
  - If the query includes GROUP BY and ORDER BY, and every column in the ORDER-BY list is also in the GROUP-BY list, there is only one sort, recorded in SORTC_ORDERBY
- Sorts to remove duplicates indicated by SORTC_UNIQ are used:
  - To process a query with SELECT DISTINCT and a function such as COUNT(DISTINCT COL1)
  - To remove duplicates in UNION processing

```
SELECT * FROM GLWGZJ.GLWTEPA
WHERE EMSTDATE > ? AND EMENDATE < ?
ORDER BY EMPTIME;
```

```
+-------------------------------------------------------------+
| PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  |
+-------------------------------------------------------------+
|   1 |   0 | GLWGZJ  | GLWTEPA | I    | GLWGZJ    | GLWXEPA2 |
|   2 |   3 |         |         |      |           |          |
+-------------------------------------------------------------+

+-----------------------------------------------------+
| PF | SNU | SNJ | SNO | SNG | SCU | SCJ | SCO | SCG |
+-----------------------------------------------------+
| L  | N   | N   | N   | N   | N   | N   | N   | N   |
|    | N   | N   | N   | N   | N   | N   | Y   | N   |
+-----------------------------------------------------+
```

# Sort And OPEN CURSOR

- If no sorts are required, OPEN CURSOR does not access any data
  - Data is accessed by Db2 – and then returned to the application – at the first fetch
- However, if a sort is required, then OPEN CURSOR causes a materialized result table to be produced
  - Control returns to the application after the result table is materialized
  - If a cursor that requires a sort is closed and reopened, the sort is performed again
- If a RID sort is performed, but no data sort, then the RID list is built from the index when the first row is fetched the first row is returned
  - Subsequent fetches access the RID pool to return the next row