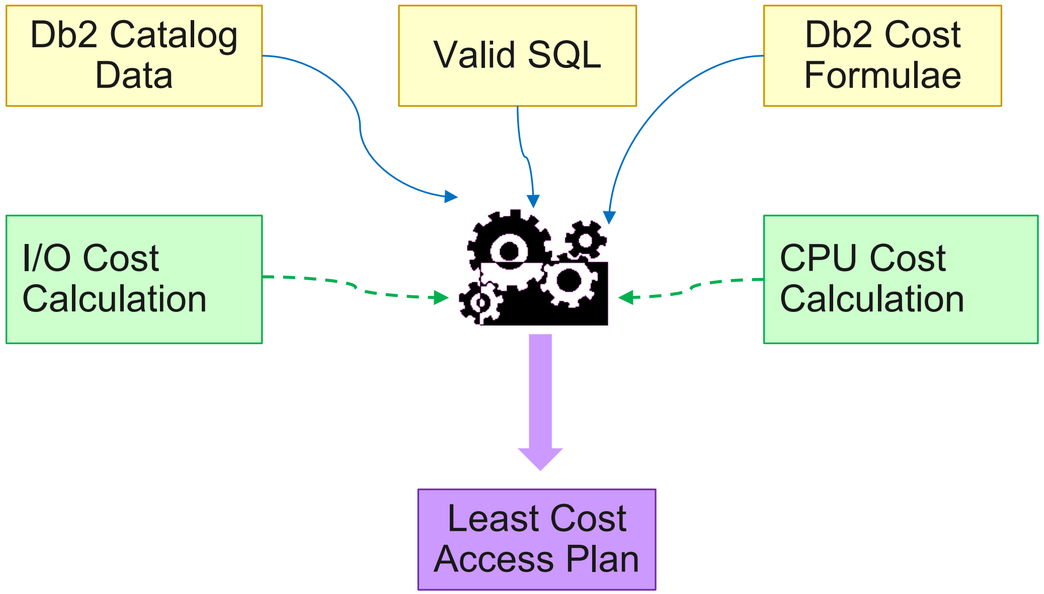
LDUG 2nd September 2026

Access Paths for Beginners

Gareth Copplestone-Jones
gcj@triton.co.uk

What are Access Paths / Access Plans

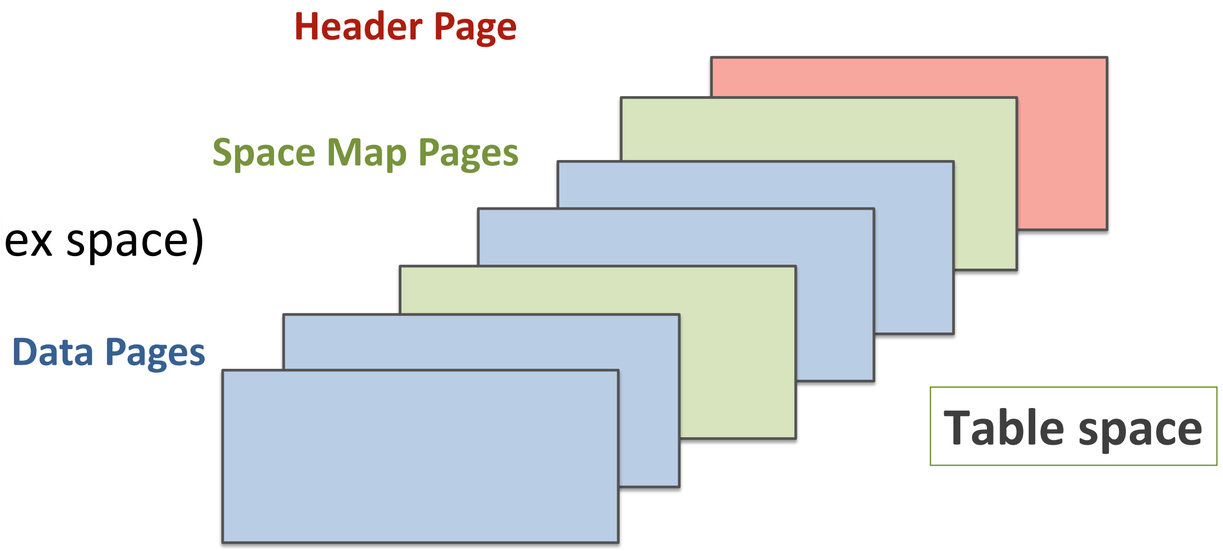
- DB2 is an automatic navigation database – you tell Db2 what data you want and Db2 builds an *access plan* or *access path* that specifies how the data is to be retrieved
  - Calculated at BIND time for static SQL, at PREPARE time for dynamic SQL
- DB2 calculates the least-cost access plan using:
  - Db2 metadata about the tables and any indexes on them
  - Statistics about the number of rows, data distribution, predicates, etc.
  - Internal DB2 cost formulae for I/O and CPU resource

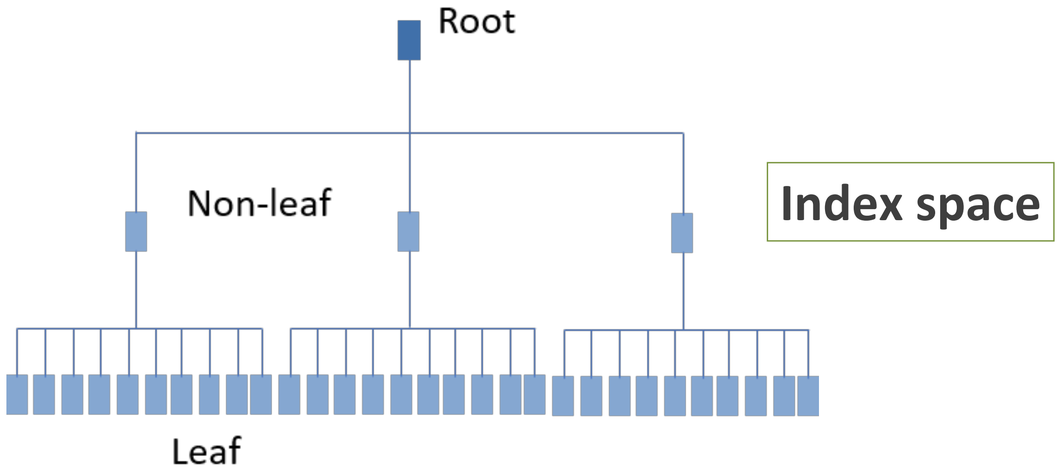


- Information about the access plan can be found in the EXPLAIN tables

Cost Calculations

- CPU cost calculation:
  - Traversing rows and pages (table space and index space)
  - Applying predicates
  - Sorting
- I/O cost calculation
  - Table space scan
  - Index scan
  - Index probe
  - Workfile scan





DB2 GETPAGE and I/O Operations

- GETPAGE operations
  - DB2 performs a GETPAGE operation whenever it needs to access a data page or index page
  - Either *random* or *sequential*
    - Random – direct access, e.g. single page access to retrieve a single row
    - Sequential – as the name implies, to process rows sequentially or skip-sequentially
  - If the page is not in the local buffer pool, DB2 performs an I/O operation to bring the page into the buffer pool
- I/O operations
  - Synchronous
    - Typically to satisfy random GETPAGE operations
    - The application waits while the page is read into the local buffer pool from disk
    - I/O is one page at a time
  - Asynchronous or *prefetch* I/O
    - These typically to satisfy sequential and skip-sequential GETPAGE operations
    - Read ahead of a set of pages into the buffer pool with only one I/O operation

Single Table Access Paths – Tablespace Scan

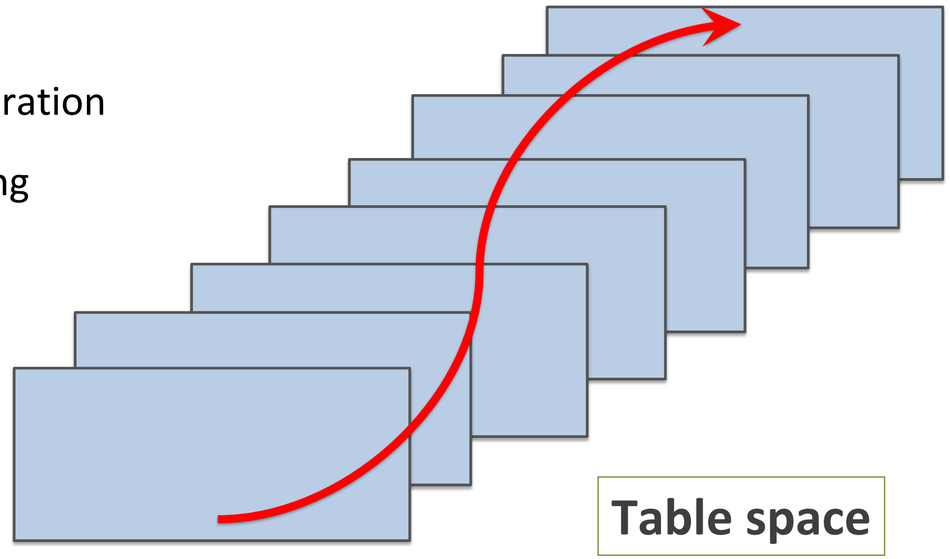
- Tablespace scan – Db2 reads the entire tablespace – for example, because:
  - Using an index is not possible because no index is available, or because none of the predicates match the columns in an available index
  - The table is small
  - A large proportion of the rows in the table qualify, therefore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

```
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  |
+-----------------------------------------------------------+
```

More About Prefetch

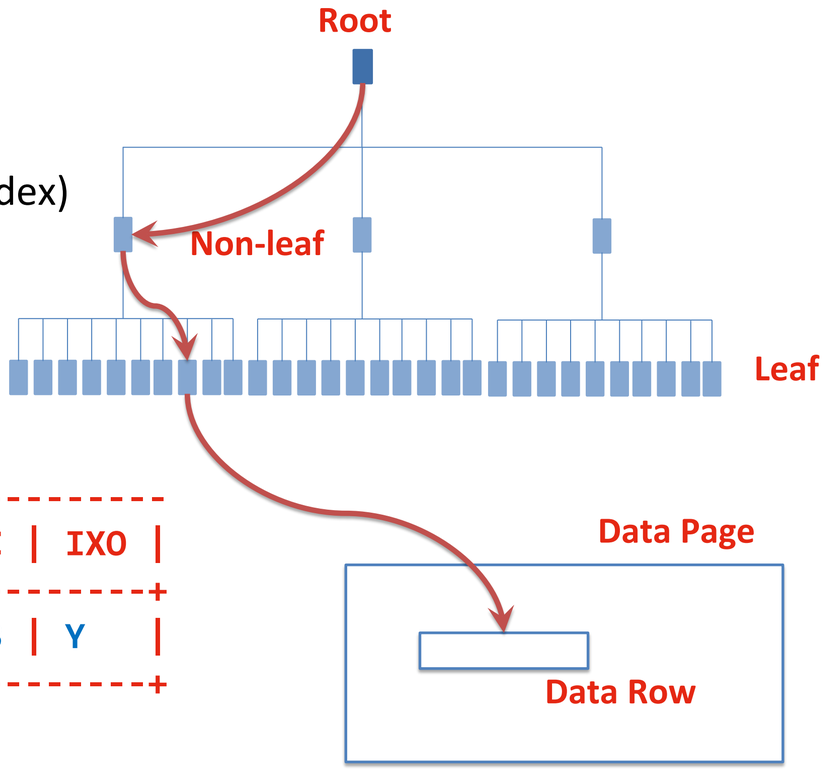
- Prefetch reads a set of pages into the buffer pool with one asynchronous I/O operation
  - Prefetch can provide substantial elapsed time, CPU and I/O savings by avoiding synchronous read I/O operations
- There are three kinds of prefetch used by DB2:
- Dynamic prefetch
  - DB2 uses a sequential detection algorithm to determine whether data pages accessed via an index are being read sequentially
  - DB2 also uses the algorithm to determine if index leaf pages are being read in key-sequential order
    - More likely for an organized index
    - Synchronous I/O still occurs for the non-leaf pages
- Sequential prefetch
  - Exclusively used for table space scans
- List prefetch – more later



Single Table Access Paths – Index Probe

- DB2 uses an index to retrieve the row requested by the application:
  - ACCESSTYPE = 'I' (index access)
  - MATCHCOLS > 0 (predicates are specified on one or more leading columns of the index)

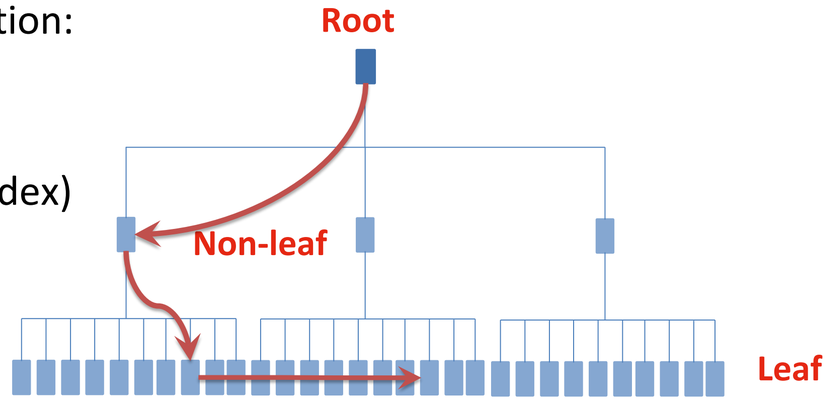
```
SELECT LASTNAME, FIRSTNME, EMP_NO
FROM GLWGZJ.GLWTEMP
WHERE LASTNAME = ? AND FIRSTNME = ? AND EMP_NO = ?;
```



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

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)



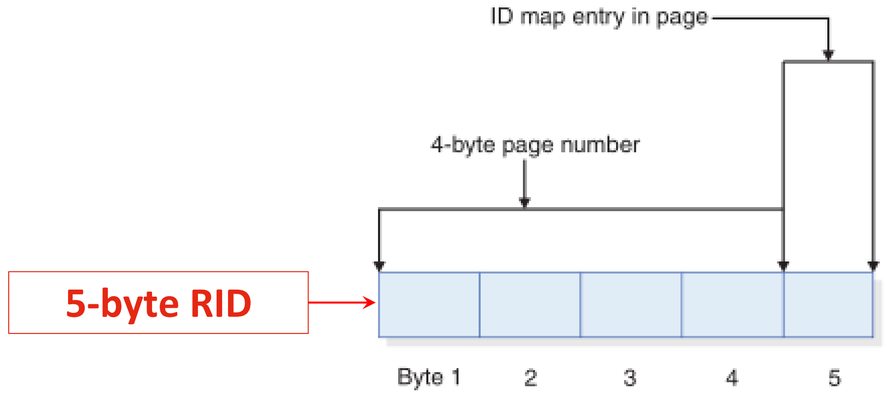

```
SELECT LASTNAME, FIRSTNME, EMP_NO
FROM GLWGZJ.GLWTEMP
WHERE LASTNAME = ? AND FIRSTNME = ? AND EMP_NO > ?;
```

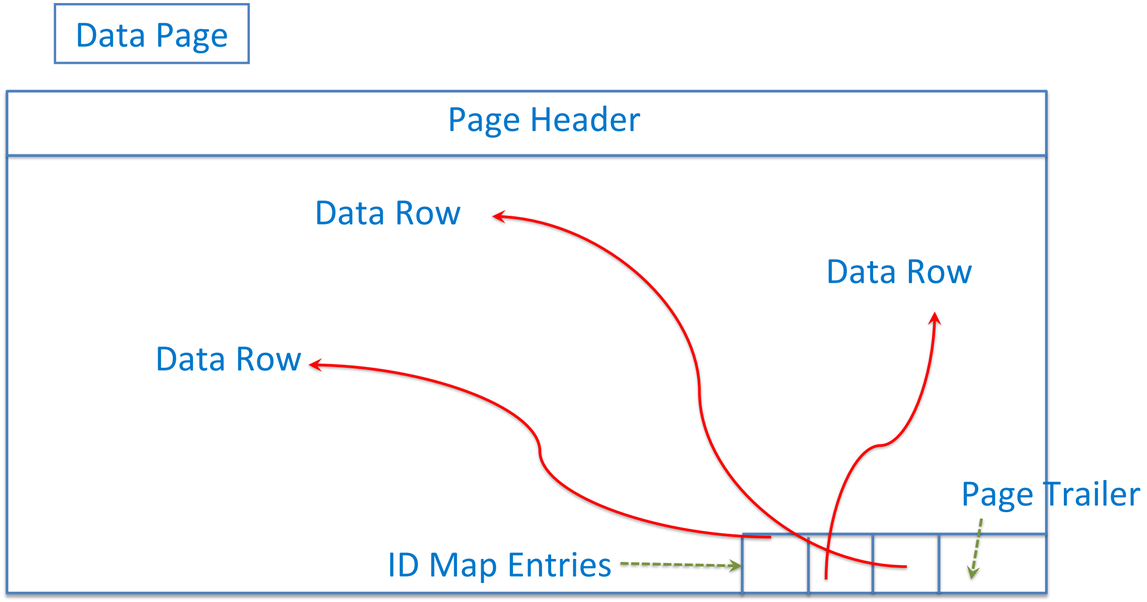
```
+-------------------------------------------------------------------------------------+
| QNO | QBN | PNO | MTH | CREATOR |  TABLE  | ATYP | IXCREATOR |  IXNAME  | MC | IXO |
+-------------------------------------------------------------------------------------+
| 777 |   1 |   1 |   0 | GLWGZJ 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

Record Identifiers (RIDs) and ID Map Entries

- A RID is a pointer composed of four (segmented, classic partitioned, simple table spaces), five (LARGE and UTS table spaces) or seven (UTS PBR RPN) bytes
  - For all but PBR RPN, the first three/four bytes contain the absolute page number from the start of the table space and the last byte the ID map entry for the row
  - PBR RPN RID contains the 2-byte partition number, the 4-byte page number relative to the start of the partition, and the 1-byte ID map entry)
- 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
- Index entries are comprised of the key and one or more RID pointer





Single Table Access Paths – IN-list Index Scan

- IN-list index access – special case of matching index scan:
  - A single indexable IN predicate is used as a matching equal predicate
  - Effectively a series of matching index scans with the values for the IN predicate 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

- IN-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

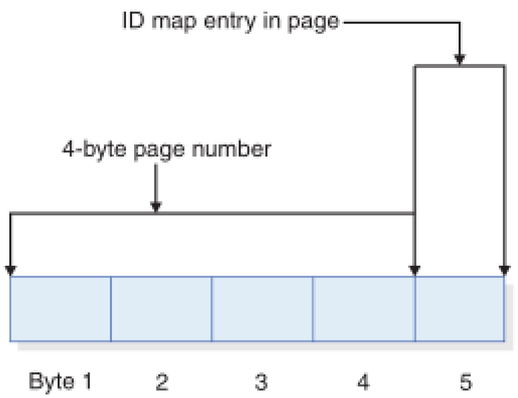
```
SELECT * FROM GLWGZJ.GLWTEMP
WHERE WORKDEPT IN (?, ?, ?) AND JOB IN (?, ?, ?);
```

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

List Prefetch

- DB2 reads index entries and 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
  - The sort is always performed in memory and is very fast
- List prefetch does not use sequential detection
- DB2 uses the RID pool to process the RID list for list prefetch
- Typically used for a multi-row result set:
  - For an index with a cluster ratio lower than 80%
  - For indexes with a high cluster ratio, if Db2 estimates the amount of data to be evaluated is too small for efficient dynamic/sequential prefetch, but still accesses multiple pages
  - Always for multiple index access (see later)
  - Always for a hybrid join (see later)
  - For updatable cursors when index columns might be updated.



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

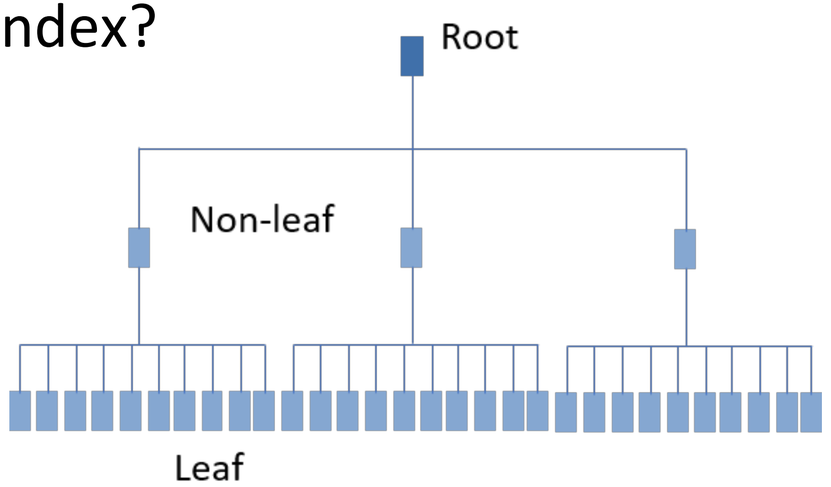
```
SELECT * FROM GLWGZJ.GLWTEMP WHERE WORKDEPT = ? AND JOB = ?;
```

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

Index Choices

- With multiple indexes, what influences the choice of index?
  - Which searched columns are indexed?
  - Matching vs non-matching index scan
  - Index only access possible?
  - The number of levels in the index
  - Cluster ratio
    - How well the order of the data rows follows the logical ordering of an index
    - Only matters for sequential / skip-sequential access
  - How well is the index organised?



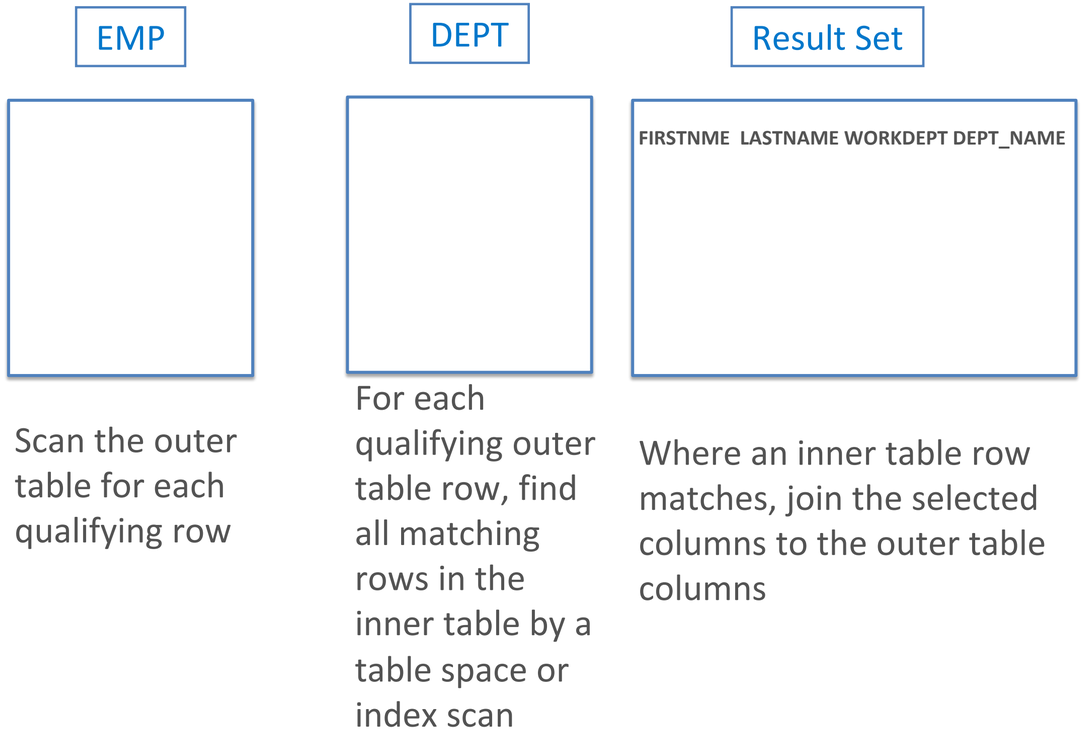
JOIN Processing

- 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)
    - A Cartesian join is a rarely used form of nested loop join where two tables in the same query have no join predicates
  - Merge scan join (METHOD = 2)
    - Also know as *sort merge join*
  - Hybrid Join (METHOD = 4)
  - Star schema access (JOIN_TYPE = 'S' or 'P') or *star join* – not covered

Nested Loop Join – METHOD = 1

- In a nested loop join, DB2:
  - Scans the *composite* (outer) table.
  - For each qualifying row (matching any *local* predicates), DB2 searches for matching rows of the *new* (inner) table
- More likely to be used when one or more of the following is true:
  - The outer table is small
  - Efficient predicate filtering limits the number of qualifying rows in the outer table
  - 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:

```
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 ;
```



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   |
+-----------------------------------------------+
```

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
    - For joins, 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 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 recorded in column SORTC_UNIQ and 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					

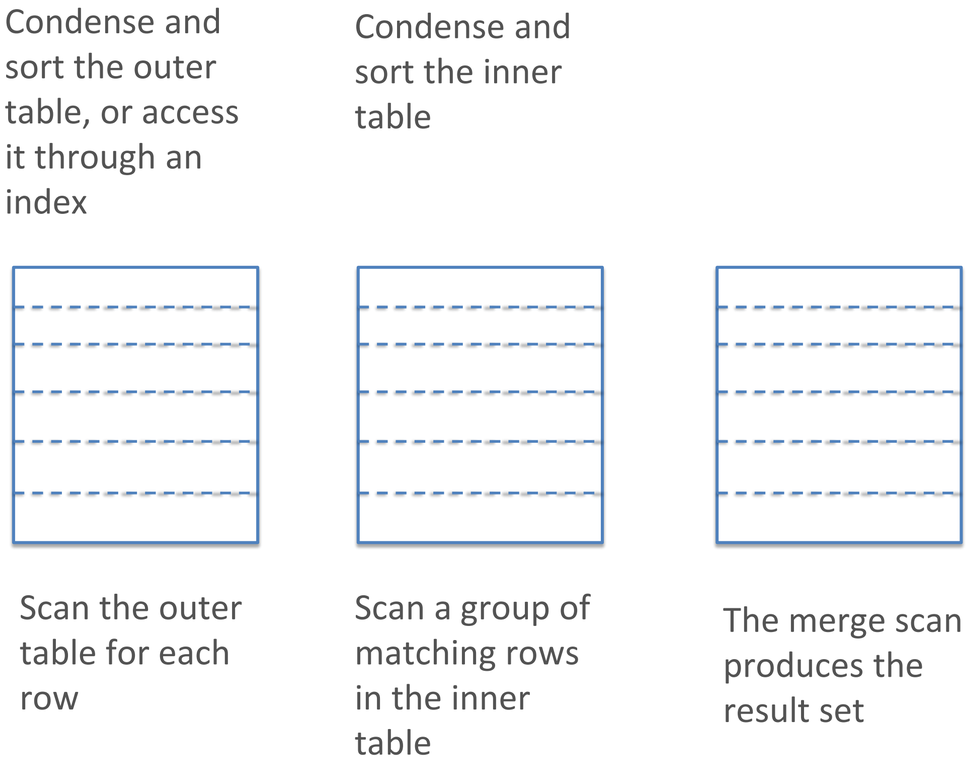
PF	SNU	SNJ	SNO	SNG	SCU	SCJ	SCO	SCG
L	N	N	N	N	N	N	N	N
	N	N	N	N	N	N	Y	N

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

Merge Scan Join – METHOD = 2

- There must be join predicates on the two tables having 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 column, 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
- More likely to be used when:
  - The number of qualifying rows in the inner and outer table is large, and the join predicates do not provide much filtering – 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

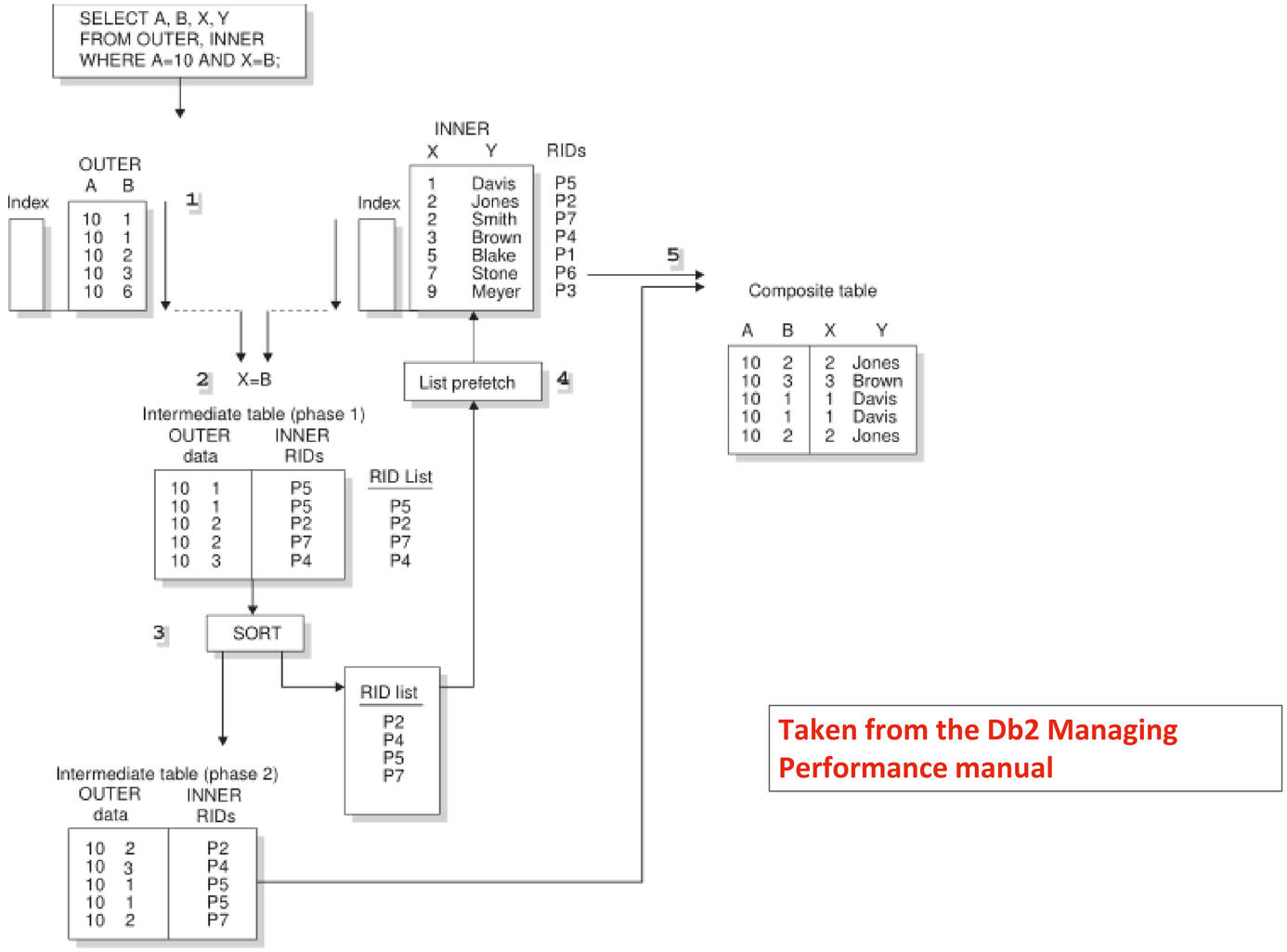
```
SELECT PJA.PROJ_NO, PJA.ACT_NO, PJA.ACTDESC,
EPA.EMP_NO
FROM GLWGZJ.GLWTPJA PJA, GLWGZJ.GLWTEPA EPA
WHERE PJA.ACT_NO = EPA.ACT_NO
AND EPA.EMENDATE < PJA.ACENDATE
ORDER BY PJA.PROJ_NO ;
```

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 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 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   |
+-------------------------------------------------+
```

Outer Joins

- An outer join keeps unmatched rows for one or both 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 |
+-----------------------------------------------------------------+
```

Subqueries, View and Nested Table Expression Materialisation

- 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
    - When a subquery is transformed into a join, the PLAN_TABLE rows for the transformed subquery have the same QBLOCKNO as the outer query

```
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 (?, ?, ?, ?, ?, ?)
)) ;
```

- 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’

```
SELECT X.C1 FROM
(SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
ON X.C2=Y.C2;;
```

Query Blocks

- When an SQL statement contains more than one subquery, the different subqueries are likely to be accessed in multiple *query blocks*
  - There are two kinds of subqueries, non-correlated subqueries and correlated subqueries:
    - Non-correlated subqueries – the inner subquery doesn't contain a reference to the outer 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 (?, ?, ?, ?, ?, ?)) ;
```

    - Correlated subqueries – the inner subquery contains a reference to the outer 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 ) ;
```

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

Thank You

LDUG 2nd September 2026

Access Paths for Beginners

Gareth Copplestone-Jones
gcj@triton.co.uk

LDUG 2nd September 2026

Access Paths for Beginners Appendix

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   |
+-----------------------------------------------------------------------------------------+
```

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