SQL Performance and Tuning

DB2 Relational Database

June 2002

Penny Bowman and Rick McClendon

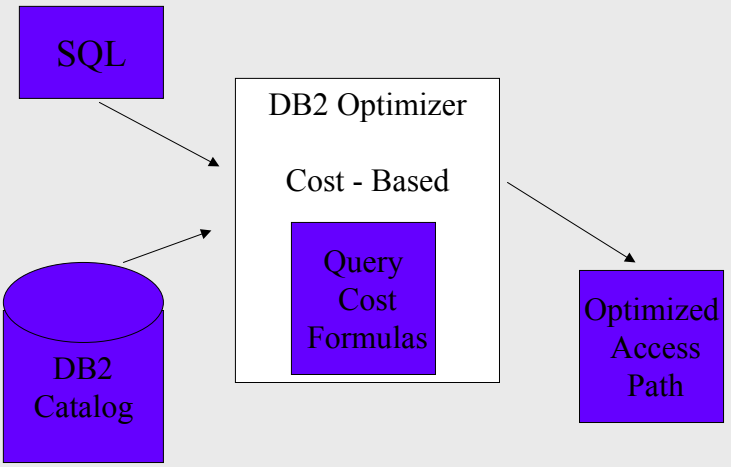
Course Overview

- The DB2 Optimizer
- SQL Coding Strategies and Guidelines
- DB2 Catalog
- Filter Factors for Predicates
- Runstats and Reorg Utilities
- DB2 Explain
- DB2 Insight

The DB2 Optimizer

- Determines database navigation
- Parses SQL statements for tables and columns which must be accessed
- Queries statistics from DB2 Catalog (populated by RUNSTATS utility)
- Determines least expensive access path
- Since it is a **Cost-Based Optimizer** - it chooses the lease expensive access path

The DB2 Optimizer



Optimizer Access Path Selection

- 1. Gets the current statistics from **DB2 catalog** for the columns and tables identified in the SQL statements. These statistics are populated by the **Runstats** utility.
- 2. Computes the estimated percentage of qualified rows for each predicate - which becomes the **filter factor** for the predicate.

Optimizer Access Path Selection

- 3. Chooses a **set** of reasonable **access paths**.
- 4. Computes each potential access path's **estimated cost** based on:
  - CPU Cost
  - I/O Cost

Access Path Cost Based On:

- CPU Cost
  - Applying predicates (Stage 1 or Stage 2)
  - Traversing pages (index and tablespace)
  - Sorting
- I/O Cost
  - DB2 Catalog statistics
  - Size of the bufferpools
  - Cost of work files used (sorts, intermediate results, and so on)

Will a Scan or an Index Be Used?

A tablespace Scan sequentially reads all of the tablespace pages for the table being accessed.

Most of the time, the fastest way to access DB2 data is with an Index. For DB2 to consider using an index - the following criteria must be met:

- At least one of the predicates for the SQL statement must be indexable.
- One of the columns (in any indexable predicate) must exist as a column in an available index.

Will a Scan or an Index Be Used?

An index will **not** be used in these circumstances:

- When no indexes exist for the table and columns being accessed
- When the optimizer determines that the query can be executed more efficiently without using an index -
  - the table has a small number of rows or
  - using the existing indexes might require additional I/O - based on the cardinality of the index and the cluster ratio of the index.

Types of Indexed Access

Direct Index Lookup

- values must be provided for each column in the index

Matching Index Scan (absolute positioning)

- can be used if the high order column (first column) of an index key is provided

Nonmatching Index Scan (relative positioning)

- can be used if the first column of the index is not provided
- can be used for non-clustered indexes
- can be used to maintain data in a particular order to satisfy the ORDER BY or GROUP BY

Index Only Access

- can be used with if a value is supplied for all index columns - avoids reading data pages completely

Sequential Prefetch

A read-ahead mechanism invoked to prefill DB2's buffers so that data is already in memory before it is requested. Can be requested by DB2 under any of these circumstances:

- A tablespace scan of more than one page
- An index scan in which the data is clustered and DB2 determines that eight or more pages must be accessed.
- An index-only scan in which DB2 estimates that eight or more leaf pages must be accessed.

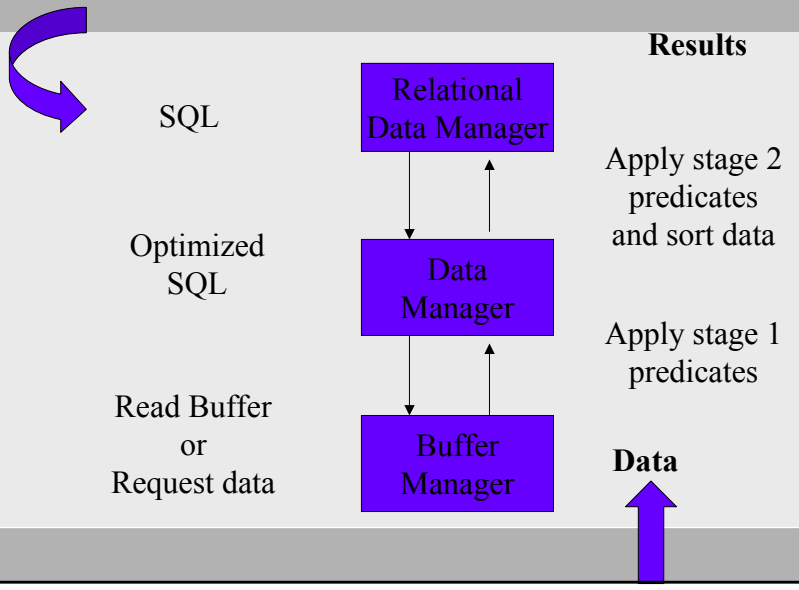
Database Services Address Space

- The DBAS, or Database Services Address Space, provides the facility for the manipulation of DB2 data structures. The DBAS consists of three components:
- Relational Data System (RDS)
  - Set-Level Orientation
  - Stage 2 predicates
  - SQL statement checking
  - Sorting
  - Optimizer

Database Services Address Space

- Data Manager (DM)
  - Row-Level Orientation
  - Stage 1 predicates
  - Indexable predicates
  - Locking
  - Various data manipulations
- Buffer Manager (BM)
  - Physical Data Access
  - Data movement to and from DASD , Bufferpools

Database Services Address Space



Database Services Address Space

- When an SQL statement requesting a set of columns and rows is passed to the RDS, the RDS determines the best mechanism for satisfying the request. The RDS can parse an SQL statement and determine its needs.
- When the RDS receives an SQL statement, it performs these steps:
- 1. Checks authorization
- 2. Resolves data element names into internal identifiers
- 3. Checks the syntax of the SQL statement
- 4. Optimizes the SQL statement and generates an access path

Database Services Address Space

- The RDS then passes the optimized SQL statement to the Data Manager for further processing.
- The function of the DM is to lower the level of data that is being operated on. The DM analyzes the request for table rows or index rows of data and then calls the Buffer Manager to satisfy the request.
- The Buffer Manager accesses data for other DB2 components. It uses pools of memory set aside for the storage of frequently accessed data to create an efficient data access environment.

Database Services Address Space

- The BM determines if the data is in the bufferpool already. If so - the BM accesses the data and send it to the DM. If not - it calls the VSAM Media Manager to read and send back the data to the BM, so it can be sent to the DM.
- The DM receives the data and applies as many predicates as possible to reduce the answer set. Only Stage 1 predicates are applied in the DM.

Database Services Address Space

- Finally, the RDS receives the data from the DM. All Stage 2 predicates are applied, the necessary sorting is performed, and the results are returned to the requestor.
- Considering these steps, realize **that Stage 1 predicates are more efficient** because they are evaluated earlier in the process, by the DM instead of the RDS, and thereby reduce overhead during the processing steps.

SQL Coding Strategies and Guidelines

- Understand Stage 1 and Stage 2 Predicates
- Tune the queries that are executed more frequently first!
- It Depends!
- Know Your Data!
- Static vs. Dynamic SQL
- Batch vs. Interactive (CICS vs. web)

Unnecessary SQL

- Avoid unnecessary execution of SQL
- Consider accomplishing as much as possible with a single call, rather than multiple calls

Rows Returned

- Minimize the number of rows searched and/or returned
- Code predicates to limit the result to only the rows needed
- Avoid generic queries that do not have a WHERE clause

Column Selection

- Minimize the number of columns retrieved and/or updated
- Specify only the columns needed
- Avoid SELECT *
- Extra columns increases row size of the result set
- Retrieving very few columns can encourage index-only access

Singleton SELECT vs. Cursor

- If a single row is returned
  - Singleton SELECT .. INTO
    - outperforms a Cursor
    - error when more than 1 row is returned
- If multiple rows are returned
  - Cursor
    - requires overhead of OPEN, FETCH, and CLOSE
- What is an example of a singleton select and a select requiring a cursor?

Singleton SELECT vs. Cursor

- For Row Update:
  - When the selected row must be retrieved first:
  - Use FOR UPDATE OF clause with a CURSOR
- Using a Singleton SELECT
  - the row can be updated by another program after the singleton SELECT but before the subsequent UPDATE, causing a possible data integrity issue

Use For Fetch Only

- When a SELECT statement is used only for data retrieval - use FOR FETCH ONLY
- FOR READ ONLY clause provides the same function - and is ODBC compliant
- Enables DB2 to use ‘block fetch’
- Monitor the performance to decide which is best for each situation

Avoid Sorting

- DISTINCT - always results in a sort
- UNION - always results in a sort
- UNION ALL - does not sort, but retains any duplicates

Avoid Sorting

- ORDER BY
  - may be faster if columns are indexed
  - use it to guarantee the sequence of the data
- GROUP BY
  - specify only columns that need to be grouped
  - may be faster if the columns are indexed
  - do not include extra columns in SELECT list or GROUP BY because DB2 must sort the rows

Subselects

- DB2 processes the subselect (inner select) first before the outer select
- You may be able to improve performance of complex queries by coding a complex predicate in a subselect
- Applying the predicate in the subselect may reduce the number of rows returned

Use Inline Views

- Inline views allow the FROM clause of a SELECT statement to contain another SELECT statement
- May enhance performance of the outer select by applying predicates in the inner select
- Useful when detail and aggregated data must be returned in a single query

Indexes

- Create indexes for columns you frequently:
  - ORDER BY
  - GROUP BY (better than a DISTINCT)
  - SELECT DISTINCT
  - JOIN
- Several factors determine whether the index will be used

Avoid Data Conversion

- When comparing column values to host variables - use the same
  - Data Type
  - Length
- When DB2 must convert data, available indexes are sometimes not used

Join Predicates

- Response time -> determined mostly by the number of rows participating in the join
- Provide accurate join predicates
- Never use a JOIN without a predicate
- Join ON indexed columns
- Use Joins over subqueries

Join Predicates (cont.)

- When the results of a join must be sorted -
  - limiting the ORDER BY to columns of a single table can avoid a sort
  - specifying columns from multiple tables causes a sort
- Favor coding explicit INNER and LEFT OUT joins over RIGHT OUTER joins
  - EXPLAIN converts RIGHT to LEFT join

Example: Outer Join With A Local Predicate

```
SELECT emp.empno, emp.lastname, dept.deptname
FROM emp LEFT OUTER JOIN dept
ON emp.workdept = dept.deptno
WHERE emp.salary > 50000.00;
```

Works correctly but… the outer join is performed first, before any rows are filtered out.

Example: Outer Join Using An Inline View

```
SELECT emp.empno, emp.lastname, dept.deptname
FROM  (SELECT empno, lastname
    FROM emp WHERE salary > 50000.00) as e
LEFT OUTER JOIN dept
ON emp.workdept = dept.deptno
```

Works better… applies the inner join predicates first, reducing number of rows to be joined

OR vs. UNION

- OR requires Stage 2 processing
- Consider rewriting the query as the union of 2 SELECTs, making index access possible
- UNION ALL avoids the sort, but duplicates are included
- Monitor and EXPLAIN the query to decide which is best

Use BETWEEN

- BETWEEN is usually more efficient than <= predicate and the >= predicate
- Except when comparing a host variable to 2 columns

Stage 2 : WHERE

:hostvar BETWEEN col1 and col2

Stage 1: WHERE

Col1 <= :hostvar AND col2 >= :hostvar

Use IN Instead of Like

- If you know that only a certain number of values exist and can be put in a list
  - Use IN or BETWEEN
    - IN ('Value1', 'Value2', 'Value3')
    - BETWEEN :valuelow AND :valuehigh
  - Rather than:
    - LIKE 'Value_'

Use LIKE With Care

- Avoid the % or the _ at the beginning because it prevents DB2 from using a matching index and may cause a scan
- Use the % or the _ at the end to encourage index usage

Avoid NOT

- Predicates formed using NOT are Stage 1
- But they are not indexable
- For Subquery - when using negation logic:
  - Use NOT Exists
    - DB2 tests non-existence
  - Instead of NOT IN
    - DB2 must materialize the complete result set

Use EXISTS

- Use EXISTS to test for a condition and get a True or False returned by DB2 and not return any rows to the query:

```
SELECT col1 FROM table1
WHERE EXISTS
   (SELECT 1 FROM table2
      WHERE table2.col2 = table1.col1)
```


Code the Most Restrictive Predicate First

- After the indexes, place the predicate that will eliminate the greatest number of rows first
- Know your data
  - Race, Gender, Type of Student, Year, Term

Avoid Arithmetic in Predicates

- An index is not used for a column when the column is in an arithmetic expression.
- Stage 1 but not indexable

```
SELECT col1
FROM table1
WHERE col2 = :hostvariable + 10
```


Limit Scalar Function Usage

- Scalar functions are not indexable
- But you can use scalar functions to offload work from the application program
- Examples:
  - DATE functions
  - SUBSTR
  - CHAR
  - etc.

Other Cautions

- Predicates that contain concatenated columns are not indexable
- SELECT Count(*) can be expensive
- CASE Statement - powerful but can be expensive

With OPTIMIZE for n ROWS

- For online applications, use ‘With OPTIMIZE for n ROWS’ to attempt to influence the access path DB2 chooses
- Without this clause, DB2 chooses the best access path for batch processing
- With this clause, DB2 optimizes for quicker response for online processing
- Try Optimize for 1, for 10, for 100

Review DB2 Optimizer

- DB2 is a Cost-based optimizer
- RUNSTATS populates the DB2 Catalog
- DB2 Catalog used to determine access path
- Create Indexes for columns you frequently select and sort
- Avoid Unnecessary Sorts in SQL
- Code the SQL predicates thoughtfully

DB2 Catalog

- SYSTABLES
- SYSTABLESPACE
- SYSINDEXES
  - FIRSTKEYCARDF
- SYSCOLUMNS
  - HIGH2KEY
  - LOW2KEY
- SYSCOLDIST
- SYSCOLDISTSTATS

Filter Factors for Predicates

- Filter factor is based on the number of rows that will be filtered out by the predicate
- A ratio that estimates I/O costs
- The lower the filter factor, the lower the cost, and in general, the more efficient the query
- Review the handout as we discuss this topic

Filter Factors for DB2 Predicates

- **Filter Factor Formulas** - use FIRSTKEYCARDF column from the SYSINDEXES table of the Catalog
- If there are no statistics for the indexes, the **default filter factors** are used
- The lowest default filter factor is .01 :
  - Column BETWEEN Value1 AND Value2
  - Column LIKE 'char%'
- Equality predicates have a default filter factor of .04 :
  - Column = value
  - Column = :hostvalue
  - ColumnA = ColumnB (of different tables)
  - Column IS NULL

Filter Factors for DB2 Predicates

- Comparative Operators have a default filter factor of .33
  - Column <, <=, >, >= value
- IN List predicates have a filter factor of .04 * (list size)
  - Column IN (list of values)
- Not Equal predicates have a default filter factor of .96 :
  - Column <> value
  - Column <> :hostvalue
  - ColumnA <> ColumnB (of different tables)

Filter Factors for DB2 Predicates

- Not List predicates have a filter factor of

  1 - (.04 * (list size))
  - Column NOT IN (list of values)
- Other Not Predicates that have a default filter factor of .90
  - Column NOT BETWEEN Value1 and Value2
  - Column NOT IN (non-correlated subquery)
  - Column <> ALL (non-correlated subquery)

Column Matching

- With a composite index, the column matching stops at one predicate past the last equality predicate.
- See Example in the handout that uses a 4 column index.
- (C1 = :hostvar1 AND C2 = :hostvar2 AND C3 = (non column expression) AND C4 > :hostvar4)
  - Stage 1 - Indexable with 4 matching columns
- (C1 = :hostvar1 AND C2 BETWEEN :hostvar2 AND :hostvar3 AND C3 = :hostvar4)
  - Stage 1 - Indexable with 2 matching columns

Column Matching

- (C1 > value1 AND C2 = :hostvar2 AND C2 IN (value1, value2, value3, value4))
  - Stage 1 - Indexable with 1 matching column
- (C1 = :hostvar1 AND C2 LIKE 'ab%xyz_1' AND C3 NOT BETWEEN :hostvar3 AND :hostvar4 AND C4 = value1)
  - Indexable with C1 = :hostvar1 AND C2 LIKE 'ab%xyz_1'
  - Stage 1 - LIKE 'ab%xyz_1' AND C3 NOT BETWEEN :hostvar3 AND :hostvar4 AND C4 = value1

Column Matching - 2 Indexes

- With two indexes: C1.C2 and C3.C4
- (C1 = :hostvar1 AND C2 LIKE :hostvar2) OR
- (C3 = (non column expression) AND C4 > :hostvar4)
  - Multiple Index Access
  - 1 column matching of first index
  - 2 columns matching on second index
  - LIKE will be Stage 2
  -

Order of Predicate Evaluation

1. Indexed predicates
2. Non-indexed predicates - Stage 1 then Stage 2

Within each of the groups above, predicates are evaluated in this sequence:

1. Equality predicates, including single element IN list predicates
2. Range and NOT NULL predicates
3. All other predicates

If multiple predicates are of the exact same type, they are evaluated in the order in which they are coded in the predicate.

Review Filter Factors for Predicates

- DB2 Catalog
- Filter Factors
- Column Matching
- Order of Predicate Evaluation

Runstats and Reorg

- Runstats Utility
  - updates the catalog tables with information about the tables in your system
  - used by the Optimizer for determining the best access path for a SQL statement
- Reorg Utility
  - reorganizes the data in your tables
  - good to run the RUNSTATS after a table has been reorg'd
- Use Workbench to review the statistics in both Test and Production databases

Runstats and Reorg (DBA Tools)

- Development Databases
  - you can use Workbench to run RUNSTATS and REORGs
- Test Databases
  - use TSO MASTER Clist to copy production statistics to the Test region
  - DBA has set this up for each project
- Production Databases
  - DBA runs the REORG and RUNSTATS utilities on a scheduled basis for production tables

DB2 Explain

- Valuable monitoring tool that can help you improve the efficiency of your DB2 applications
- Parses your SQL statements and reports the access path DB2 plans to use
- Uses a **Plan Table** to contain the information about each SQL statement. Each project has their own plan table.

DB2 Explain

- Required and reviewed by DBA when a DB2 program is moved to production
- Recognizes the ? Parameter Marker - assumes same data type and length as you will define in your program
- Know your data, your table design, your indexes to maximize performance

DB2 Explain Example - 1 Table

```
--SET THE CURRENT SQLID TO YOUR AREA:
SET CURRENT SQLID='FSUDBA';

-- SET THE QUERYNO TO YOUR USERID NUMBER, OR SOMETHING UNIQUE
--    IN YOUR GROUP. IN THIS EXAMPLE, CHANGE 587 TO YOUR USERID.

--THIS QUERY SELECTS COLUMNS FROM 1 TABLE.
--NOTICE THE ? PARAMETER MARKERS IN THE WHERE CLAUSE.

EXPLAIN PLAN SET QUERYNO=587 FOR
  SELECT  NAME,  SSN,  YEAR, TERM
 FROM FSDWH.DATA_SHARE
   WHERE SUBSTR(YEAR,3,2)  = ? AND      TERM = ?;
```

DB2 Explain Example - 1 Table

```
-GENERATE THE EXPLAIN REPORT FROM THE PLAN_TABLE OF YOUR AREA:
 SELECT
    SUBSTR(DIGITS(QUERYNO),6,5) AS QUERY,
    SUBSTR(DIGITS(QBLOCKNO),4,2) AS BLOCK,
    SUBSTR(DIGITS(PLANNO),4,2) AS PLAN,
    SUBSTR(DIGITS(METHOD),4,2) AS METH,
    TNAME,   SUBSTR(DIGITS(TABNO),4,2) AS TABNO,
    ACCESSTYPE AS TYPE,  SUBSTR(DIGITS(MATCHCOLS),4,2) AS MC,
    ACCESSNAME AS ANAME, INDEXONLY AS IO,
    SORTN_UNIQ AS SNU, SORTN_JOIN AS SNJ, SORTN_ORDERBY AS SNO,
    SORTN_GROUPBY AS SNG, SORTC_UNIQ AS SCU, SORTC_JOIN AS SCJ,
    SORTC_ORDERBY AS SCO, SORTC_GROUPBY AS SCG, PREFETCH AS PF
 FROM FSUDBA.PLAN_TABLE
 WHERE QUERYNO = 587   ORDER BY 1, 2, 3;
-DELETE THE ROWS YOU ADDED DURING THIS EXPLAIN PROCESS:
 DELETE FROM FSUDBA.PLAN_TABLE   WHERE QUERYNO = 587;
```


DB2 Explain Example- 1 Table

```
---------+---------+---------+---------+---------+---------+---------+----
QUERY  BLOCK  PLAN  METH  TNAME           TABNO  TYPE  MC  ANAME
---------+---------+---------+---------+---------+---------+---------+----
00587    01       01     00      DATA_SHARE  01       I      00   IXDSH01


--+---------+---------+---------+---------+---------+-------
    IO  SNU  SNJ  SNO  SNG  SCU  SCJ  SCO  SCG  PF
--+---------+---------+---------+---------+---------+-------
    N   N    N    N    N    N    N    N    N    S
```

DB2 Explain Columns

- **QUERY Number** - Identifies the SQL statement in the PLAN_TABLE (any number you assign - the example uses the numeric part of the userid)
- **BLOCK** - query block within the query number, where 1 is the top level SELECT. Subselects, unions, materialized views, and nested table expressions will show multiple query blocks. Each QBLOCK has it's own access path.
- **PLAN** - indicates the order in which the tables will be accessed

DB2 Explain Columns

- **METHOD** - shows which JOIN technique was used:
  - 00- First table accessed, continuation of previous table accessed, or not used.
  - 01- Nested Loop Join. For each row of the present composite table, matching rows of a new table are found and joined
  - 02- Merge Scan Join. The present composite table and the new table are scanned in the order of the join columns, and matching rows are joined.
  - 03- Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, a quantified predicate, or an IN predicate. This step does not access a new table.
  - 04- Hybrid Join. The current composite table is scanned in the order of the join-column rows of the new table. The new table accessed using list prefetch.
- **TNAME** - name of the table whose access this row refers to. Either a table in the FROM clause, or a materialized VIEW name.
- **TABNO** - the original position of the table name in the FROM clause

DB2 Explain Columns

- **TYPE (ACCESS TYPE)** - indicates whether an index was chosen:
  - I = INDEX
  - R= TABLESPACE SCAN (reads every data page of the table once)
  - I1= ONE-FETCH INDEX SCAN
  - N= INDEX USING IN LIST
  - M= MULTIPLE INDEX SCAN
  - MX = NAMES ONE OF INDEXES USED
  - MI = INTERSECT MULT. INDEXES
  - MU = UNION MULT. INDEXES

DB2 Explain Columns

- **MC (MATCHCOLS)** - number of columns of matching index scan
- **ANAME (ACCESS NAME)** - name of index
- **IO (INDEX ONLY)** - Y = index alone satisfies data request
- N = table must be accessed also
- 8 Sort Groups: Each **sort group** has four indicators indicating why the sort is necessary. Usually, a sort will cause the statement to run longer.
  - UNIQ - DISTINCT option or UNION was part of the query or IN list for subselect
  - JOIN - sort for Join
  - ORDERBY - order by option was part of the query
  - GROUPBY - group by option was part of the query

DB2 Explain Columns

Sort flags for 'new' (inner) tables:

- SNU - SORTN_UNIQ - Y = remove duplicates, N = no sort
- SNJ - SORTN_JOIN - Y = sort table for join, N = no sort
- SNO - SORTN_ORDERBY - Y = sort for order by, N = no sort
- SNG - SORTN_GROUPBY - Y = sort for group by, N = no sort

DB2 Explain Columns

- Sort flags for 'composite' (outer) tables:
  - SCU - SORTC_UNIQ - Y = remove duplicates, N = no sort
  - SCJ - SORTC_JOIN - Y = sort table for join, N = no sort
  - SCO - SORTC_ORDERBY - Y = sort for order by, N = no sort
  - SCG - SORTC_GROUPBY - Y = sort for group by, N = no sort
  - PF - PREFETCH - Indicates whether data pages were read in advance by prefetch.
  - S = pure sequential PREFETCH
  - L = PREFETCH through a RID list
  - Blank = unknown, or not applicable

DB2 Explain Analysis

Guidelines:

- You want to avoid tablespace scans (TYPE = R) or at least be able to explain why. Tablespace scans are acceptable for small tables.
- Nested Loop Join is usually the most efficient join method.
- Index only access is desirable (but usually not possible)
- You should strive for Index access with the matching columns being the same as the number of columns in the index.

DB2 Explain Analysis

Try to answer the following questions:

- Is Access through an Index? (TYPE is I, I1, N or MX)
- Is Access through More than one Index (TYPE is M, MX, MI or MU)
- How many columns of the index are used in matching (TYPE is I, I1, N or MX and MC contains number of matching columns)
- Is the query satisfied using only the index? (IO = Y)

DB2 Explain Analysis

- Is a view materialized into a work file? (TNAME names a view)
- What Kind of Prefetching is done? (PF is L for List, S for sequential or blank)
- Are Sorts Performed? (SNU,SNJ,SNO,SNG,SCU,SCJ,SCO or SCG = Y)
- Is a subquery transformed into a join? (BLOCK Value)

DB2 Explain Example - 5 tables

This example uses the training database tables:

--SET THE CURRENT SQLID TO YOUR AREA:

SET CURRENT SQLID='FSUTRN';

-- SET THE QUERYNO TO YOUR USERID NUMBER, OR SOMETHING UNIQUE IN YOUR GROUP. IN THIS EXAMPLE, CHANGE 587 TO YOUR USERID.

--THIS QUERY SELECTS COLUMNS FROM 5 TABLES.

--NOTICE THE ? PARAMETER MARKERS IN THE WHERE CLAUSE.

EXPLAIN PLAN SET QUERYNO=587 FOR

DB2 Explain Example - 5 tables

```
SELECT C.COURSE_NUMBER, C.COURSE_IND,
    C.YEAR, C.TERM, C.SECTION_NUMBER,
    C.SUMMER_SESSION_IND, C.FACULTY_ID,
    E.COURSE_DEPT_NUMBER,
    D.LAST_NAME AS FACULTY_LAST_NAME,
    D.FIRST_NAME AS FACULTY_FIRST_NAME,
    D.MIDDLE_NAME AS FACULTY_MID_NAME,
    A.STUDENT_ID,  A.HOURS,
     B.LAST_NAME AS STUDENT_LAST_NAME,
     B.FIRST_NAME AS STUDENT_FIRST_NAME,
     B.MIDDLE_NAME AS STUDENT_MID_NAME,
     B.CURR_CLASS,  B.CURR_DIV,  B.CURR_MAJOR, B.RACE,
    B.GENDER
```


DB2 Explain Example - 5 tables

```
FROM FSDBA.COURSE_MASTER AS E,
    FSDBA.CURRENT_COURSES AS C,
    FSDBA.TEACHER_MASTER  AS D,
    FSDBA.STUDENT_COURSE  AS A,
    FSDBA.STUDENT_MASTER  AS B
WHERE
     C.COURSE_NUMBER = E.COURSE_NUMBER AND
     C.COURSE_IND    = E.COURSE_IND    AND
     C.FACULTY_ID    = D.FACULTY_ID    AND
     C.YEAR = A.YEAR   AND  C.TERM = A.TERM         AND
     C.COURSE_NUMBER = A.COURSE_NUMBER AND
     C.COURSE_IND    = A.COURSE_IND    AND
     C.SECTION_NUMBER = A.SECTION_NUMBER AND
     A.STUDENT_ID    = B.STUDENT_ID   AND
```

DB2 Explain Example - 5 tables

```
 C.YEAR = '1998' AND C.TERM = '9'  AND
STATUS NOT IN ('13', '14', '15', '20', '21') AND
-- FIRST CODE A POSSIBLE DEPT:
  (E.COURSE_DEPT_NUMBER = '1105'
  OR
-- THEN CODE THE POSSIBLE COURSE NUMBERS:
    SUBSTR(C.COURSE_NUMBER,1,7) = 'BCH4054'
  OR
-- THEN CODE THE POSSIBLE PREFIXES:
    SUBSTR(C.COURSE_NUMBER,1,3) = 'CEG'
  OR
```


DB2 Explain Example - 5 tables

```
-- THEN CODE THE POSSIBLE SECTIONS:
    ( SUBSTR(C.COURSE_NUMBER,1,7)
       LIKE  'STA4502'  AND
       SUBSTR(C.SECTION_NUMBER,1,2) LIKE '01' )
  OR  ( SUBSTR(C.COURSE_NUMBER,1,7)
       LIKE  'GEB6904'       AND
       SUBSTR(C.SECTION_NUMBER,1,2) LIKE '04' )
  OR  ( SUBSTR(C.COURSE_NUMBER,1,7)
       LIKE  'SYO5376'       AND
       SUBSTR(C.SECTION_NUMBER,1,2) LIKE '85' )
   )
```

DB2 Explain Example - 5 tables

```
ORDER,BY C.COURSE_NUMBER, C.COURSE_IND,
   C.SECTION_NUMBER,
     STUDENT_LAST_NAME,
     STUDENT_FIRST_NAME,
     STUDENT_MID_NAME
     FOR FETCH ONLY
     OPTIMIZE FOR 15 ROWS;
```


DB2 Explain Example - 5 tables

QUERY	BLOCK	PLAN	METH	TNAME	TABNO	TYPE	MC	ANAME
00587	01	01	00	CURRENT_COURSES	02	R	00	
00587	01	02	04	COURSE_MASTER	01	I	02	IXCRM01
00587	01	03	04	STUDENT_COURSE	04	I	03	IXSTC02
00587	01	04	01	TEACHER_MASTER	03	I	01	IXTCM01
00587	01	05	01	STUDENT_MASTER	05	I	01	IXSTM01
00587	01	06	03		00		00	

DB2 Explain Example - 5 tables

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

Example: SAMAS Query1

```
SET CURRENT SQLID='FSUDBA';
   EXPLAIN PLAN SET QUERYNO=587 FOR
   SELECT MON, SUM(AMOUNT)
FROM
(SELECT
  MACH_DATE,   MONTH(MACH_DATE) AS MON,
    SUM (AMOUNT) AS AMOUNT
FROM
  FSUDWH.SAMAS_TRANSACTIONS SAM,
  FSUDWH.FUND_CODES FND,
  FSUDWH.OBJECT_CODES OBJ,
  FSUDWH.APPRO_CATEGORY_CDS CAT
```

Example: SAMAS Query1

```
WHERE  ( CAT.APPRO_CATEGORY=
   SAM.APPRO_CATEGORY  )   AND
( OBJ.OBJECT_CODE=    SAM.CHARGE_OBJECT  )
  AND  ( SAM.STATE_FUND= FND.STATE_FUND AND
   SAM.FUND_ID= FND.FUND_CODE  )
  AND  (     ( SAM.RECORD_TYPE  = 'I'  )
  AND  SAM.CHARGE_ORG   LIKE  '021000000‘
  AND  SAM.MACH_DATE  BETWEEN  '2000-07-01'
  AND '2001-06-30‘  AND  ( SAM.B_D_E_R = 'D‘   AND
   SAM.TRANS_TYPE <> '80‘  AND
   SAM.RECORD_TYPE = 'I'  )   )
 GROUP BY SAM.MACH_DATE    ) AS QRY1
 GROUP BY MON ;
```


Example: SAMAS Query1

```
SELECT  SUBSTR(DIGITS(QUERYNO),6,5) AS QUERY,
        SUBSTR(DIGITS(QBLOCKNO),4,2) AS BLOCK,
        SUBSTR(DIGITS(PLANNO),4,2) AS PLAN,
        SUBSTR(DIGITS(METHOD),4,2) AS METH,
        TNAME,   SUBSTR(DIGITS(TABNO),4,2) AS TABNO,
        ACCESSTYPE AS TYPE,
     SUBSTR(DIGITS(MATCHCOLS),4,2) AS MC,
        ACCESSNAME AS ANAME, INDEXONLY AS IO,
        SORTN_UNIQ AS SNU, SORTN_JOIN AS SNJ,
     SORTN_ORDERBY AS SNO,
        SORTN_GROUPBY AS SNG, SORTC_UNIQ AS SCU,
     SORTC_JOIN AS SCJ,
        SORTC_ORDERBY AS SCO, SORTC_GROUPBY AS SCG,
     PREFETCH AS PF
      FROM FSUDBA.PLAN_TABLE
     WHERE QUERYNO = 587   ORDER BY 1, 2, 3;
     delete from fsudba.plan_table where queryno = 587;
```

Example: SAMAS Query1

QUERY	BLOCK	PLAN	METH	TNAME	TABNO	TYPE	MC	ANAME
00587	01	01	00	QRY1	01	R	00	
00587	01	02	03		00		00	
00587	02	01	00	SAMAS_TRANSACTIONS	02	I	01	IXSTR08
00587	02	02	01	FUND_CODES	03	I	02	IXFUN01
00587	02	03	01	OBJECT_CODES	04	I	01	IXOBJ01
00587	02	04	01	APPRO_CATEGORY_CDS	05	I	01	IXACC01
00587	02	05	03		00		00	

Example: SAMAS Query1

IO	SNU	SNJ	SNO	SNG	SCU	SCJ	SCO	SCG	PF
N	N	N	N	N	N	N	N	N	S
N	N	N	N	N	N	N	N	Y	
N	N	N	N	N	N	N	N	N	S
Y	N	N	N	N	N	N	N	N	
Y	N	N	N	N	N	N	N	N	
Y	N	N	N	N	N	N	N	N	
N	N	N	N	N	N	N	N	Y	

Example: SAMAS Query2

```
SET CURRENT SQLID = 'FSUDBA';
EXPLAIN PLAN SET QUERYNO = 1 FOR
   SELECT MACH_DATE, FISCAL_YEAR,
   CHARGE_ORG, PRIMARY_DOC_NUM,
   AMOUNT
FROM FSDBA.SAMAS_TRANSACTIONS
   WHERE MACH_DATE >= '2001-07-01'
   AND MACH_DATE <= '2002-06-30'
   AND FISCAL_YEAR IN  ('20012002'
   ,'20012002')
   AND BUDGET_ENTITY = '48900100'
```


Example: SAMAS Query2

```
AND APPRO_CATEGORY = '010000'
AND CERTIFY_FORWARD = ' '
   AND GL LIKE '7%'
   AND (SUBSTR(DATE_TAG,1,2)) ¬= '  '
   AND (SUBSTR(DATE_TAG,1,2)) IN
   ('01','02','03','04','05','06','07','08','09','10','11','12')
   AND PRIMARY_DOC_NUM LIKE '_OT%'
   ORDER BY MACH_DATE, FISCAL_YEAR
   FOR FETCH ONLY
   ;
```

Example: SAMAS Query2

```
SELECT SUBSTR(DIGITS(QUERYNO),6,5) AS QUERY,
    SUBSTR(DIGITS(QBLOCKNO),4,2) AS BLOCK,
    SUBSTR(DIGITS(PLANNO),4,2) AS PLAN,
    SUBSTR(DIGITS(METHOD),4,2) AS METH,
    TNAME, SUBSTR(DIGITS(TABNO),4,2) AS TABNO,
    ACCESSTYPE AS TYPE, SUBSTR(DIGITS(MATCHCOLS),4,2)
    AS MC, ACCESSNAME AS ANAME, INDEXONLY AS IO,
    SORTN_UNIQ AS SNU, SORTN_JOIN AS SNJ,
    SORTN_ORDERBY AS SNO,
    SORTN_GROUPBY AS SNG, SORTC_UNIQ AS SCU,
    SORTC_JOIN AS SCJ, SORTC_ORDERBY AS SCO,
    SORTC_GROUPBY AS SCG, PREFETCH AS PF
    FROM FSUDBA.PLAN_TABLE
    WHERE QUERYNO = 1 ORDER BY 1, 2, 3;
DELETE FROM FSUDBA.PLAN_TABLE WHERE QUERYNO = 1;
```


Example: SAMAS Query2

```
QUERY BLOCK PLAN METH TNAME              TABNO TYPE MC ANAME
----- ----- ---- ---- ------------------ ----- ---- -- ------
00001 01    01   00   SAMAS_TRANSACTIONS 01    R    00
00001 01    02   03                      00         00


IO SNU SNJ SNO SNG SCU SCJ SCO SCG PF
-- --- --- --- --- --- --- --- --- --
 N  N   N   N   N   N   N   N   N   S
 N  N   N   N   N   N   N   Y   N
```

DB2 Insight

- Use DB2 Insight to determine how much CPU is used by your query
- You can look at information during and after your query executes
- Demo
- Goal --> REDUCE the COST !!!

Review Performance Tuning

- Write your SQL to maximize use of Indexes and Stage 1 Predicates
- Use an EXPLAIN Report to understand how DB2 plans to access the data
- Run REORGs and RUNSTATs as needed
- Use Insight for obtaining actual CPU costs