May 6-10, 2007

San Jose Convention Center

San Jose, California, USA

Session: G04 & G12

Solutions for SQL Gone Wild

IDUG® 2007 North America

Sheryl M. Larsen

SMLI

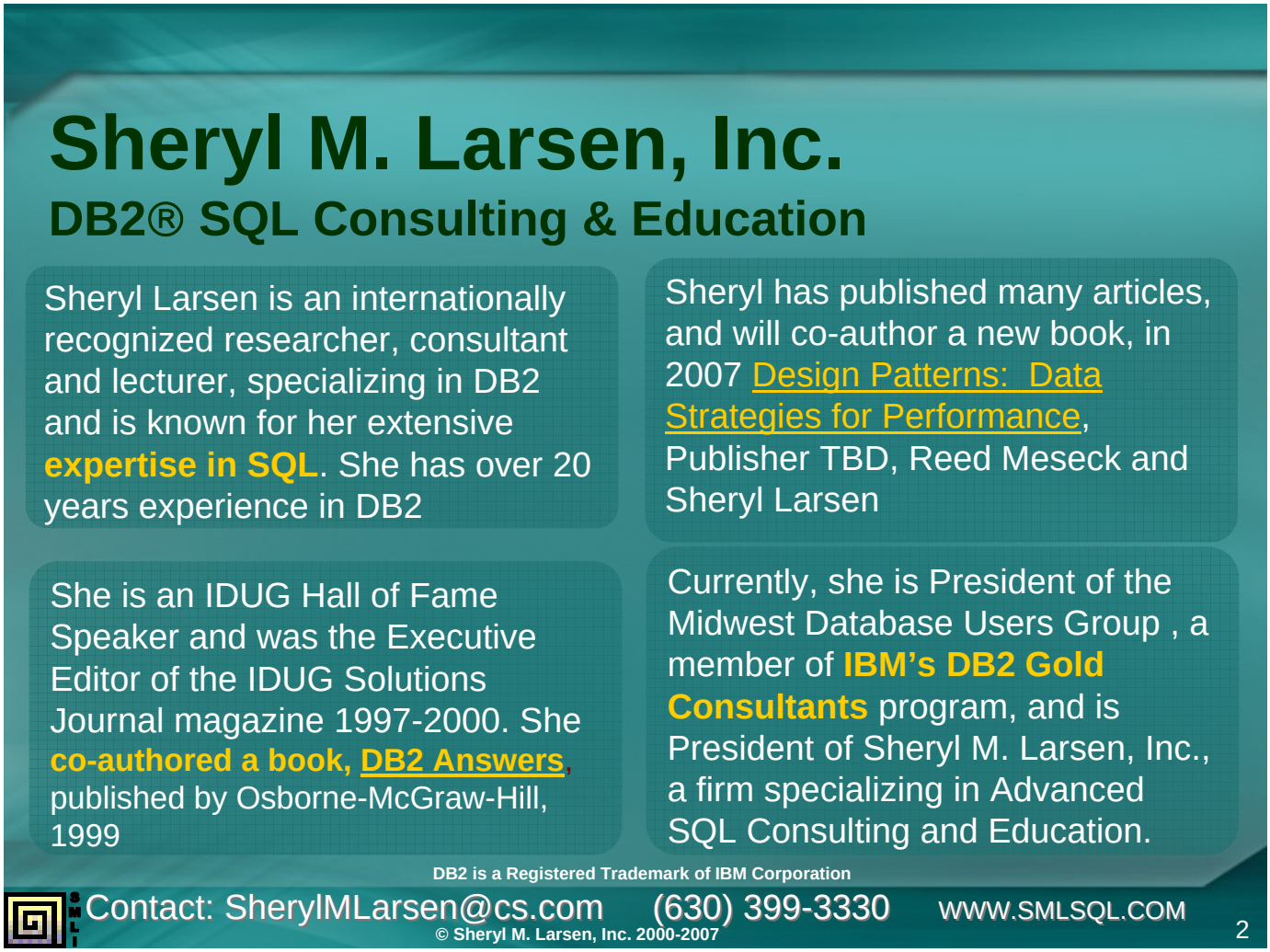
May 07, 2007 4:30 p.m. – 5:30 p.m.
May 10, 2007 9:00 a.m. – 10:00 a.m.

Platform: z/0S

GoFurther

Notes:

SQL is the nucleolus of every relational DBMS. Sometimes applications are designed with the specific custom queries; sometimes the queries are generated by a tool and sometimes a mixture of the two. SQL performance is independent of the design source or is it? Over time, SQL performance can become unpredictable. Come see examples of SQL gone wild. Solutions for taming and preventing the phenomena will be provided.



Notes:

Sheryl Larsen is an internationally recognized researcher, consultant and lecturer, specializing in DB2 and is known for her extensive expertise in SQL. Sheryl has over 20 years experience in DB2, has published many articles, several popular DB2 posters, and co-authored a book, DB2 Answers. She was voted Best Overall Speaker at the 1999 & 2001 IDUG conferences and was the Executive Editor of the *IDUG Solutions Journal* magazine 1997-2000. Currently, she is President of the Midwest Database Users Group (M_DUG), a member of IBM's DB2 Gold Consultants program, and is President of Sheryl M. Larsen, Inc., a firm specializing in Advanced SQL Consulting and Education

Sheryl M. Larsen, Inc.		Telephone:	(630) 399-3330
650 Saylor Avenue		Fax:	(630) 834-0069
Elmhurst, IL 60126		Web:	www.smlsql.com

Email: SherylMLarsen@cs.com

Agenda

- **Objective1:** Short history of SQL
- **Objective2:** Identify characteristics of SQL gone wild
- **Objective3:** Identify sources of SQL gone wild
- **Objective4:** Share solutions for taming SQL
- **Objective5:** Share solutions for preventing SQL going wild

Notes:

Short History of SQL

z/OS and LUW

Notes:

History of SQL Clauses

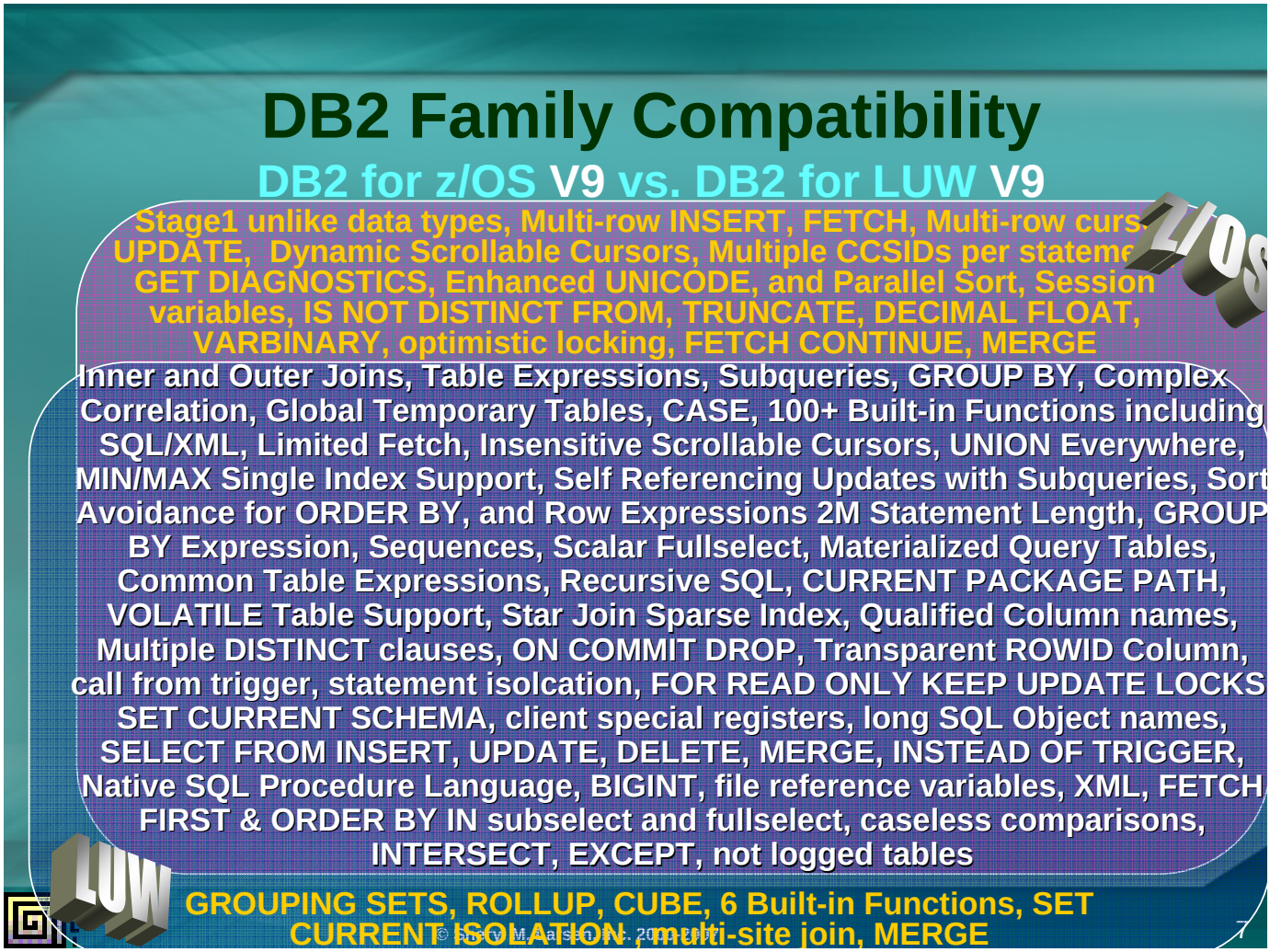
- V1 – V6 SELECT Features Uses & Examples
  - » Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, and 100+ Built-in Functions
- V7 SQL Enhancements
  - » Limited Fetch, Scrollable Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions
- V8 SQL Enhancements
  - » Stage1 unlike data types, 2M Statement Length, GROUP BY Expression, Multi-row INSERT, Multi-row FETCH, Sequences, Dynamic Scrollable Cursors, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, INSERT with SELECT, GET DIAGNOSTICS, CURRENT PACKAGE PATH, Multiple CCSIDs per statement, VOLATILE Table Support, Enhanced UNICODE, Star Join Sparse Index, Parallel Sort, Qualified Column names, Multiple DISTINCT clauses, IS NOT DISTINCT FROM, ON COMMIT DROP, Transparent ROWID Column
- Additional DB2 for LUW V8 SQL Enhancements
  - » Updateable UNION in Views, ORDER BY/FETCH FIRST in subselects & table expressions, GROUPING SETS, ROLLUP, CUBE, INSTEAD OF TRIGGER, EXCEPT, INTERSECT, and 16 Built-in Functions

5

Notes:


DB2 Family Compatibility
DB2 for LUW V8 vs. DB2 for z/OS V8
Stage1 unlike data types, Multi-row INSERT, Multi-row FETCH, Dynamic Scrollable Cursors, Multiple CCSIDs per statement, Enhanced UNICODE, and Parallel Sort
z/OS
Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions, Limited Fetch, Scrollable Cursors, UNION Everywhere, MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort Avoidance for ORDER BY, and Row Expressions 2M Statement Length, GROUP BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables, Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH, VOLATILE Table Support, Star Join Sparse Index, Qualified Column names, Multiple DISTINCT clauses, IS NOT DISTINCT FROM, ON COMMIT DROP, Transparent ROWID Column, GET DIAGNOSTICS
LUW
Updateable UNION in Views, INSERT with UPDATE/DELETE, ORDER BY/FETCH FIRST in subselects & table expressions, GROUPING SETS, ROLLUP, CUBE, INSTEAD OF TRIGGER, EXCEPT, INTERSECT, and 16 Built-in Functions
© Sheryl M. Larsen, Inc. 2000-2007
6

Notes:


DB2 Family Compatibility
DB2 for z/OS V9 vs. DB2 for LUW V9
Stage1 unlike data types, Multi-row INSERT, FETCH, Multi-row curs
UPDATE, Dynamic Scrollable Cursors, Multiple CCSIDs per stateme
GET DIAGNOSTICS, Enhanced UNICODE, and Parallel Sort, Session
variables, IS NOT DISTINCT FROM, TRUNCATE, DECIMAL FLOAT,
VARBINARY, optimistic locking, FETCH CONTINUE, MERGE
Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex
Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including
SQL/XML, Limited Fetch, Insensitive Scrollable Cursors, UNION Everywhere,
MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort
Avoidance for ORDER BY, and Row Expressions 2M Statement Length, GROUP
BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables,
Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH,
VOLATILE Table Support, Star Join Sparse Index, Qualified Column names,
Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column,
call from trigger, statement isolcation, FOR READ ONLY KEEP UPDATE LOCKS
SET CURRENT SCHEMA, client special registers, long SQL Object names,
SELECT FROM INSERT, UPDATE, DELETE, MERGE, INSTEAD OF TRIGGER,
Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH
FIRST & ORDER BY IN subselect and fullselect, caseless comparisons,
INTERSECT, EXCEPT, not logged tables
GROUPING SETS, ROLLUP, CUBE, 6 Built-in Functions, SET
CURRENT ISOLATION, multi-site join, MERGE
Z/OS
LUW

Notes:


DB2 Family Compatibility
DB2 for z/OS Vlater vs. DB2 for LUW Vlater
XXXXX XXXX XXXX XXXX, XXXXX XXXX XXXXX, XXXXXX, XXXX,
XXXXXXXX XXXXXX XXXXXXX, XXXXX XXXXXX XXXX, XXXXXX
Inner and Outer Joins, Table Expressions, Subqueries, GROUP BY, Complex
Correlation, Global Temporary Tables, CASE, 100+ Built-in Functions including
SQL/XML, Limited Fetch, Insensitive Scrollable Cursors, UNION Everywhere,
MIN/MAX Single Index Support, Self Referencing Updates with Subqueries, Sort
Avoidance for ORDER BY, and Row Expressions 2M Statement Length, GROUP
BY Expression, Sequences, Scalar Fullselect, Materialized Query Tables,
Common Table Expressions, Recursive SQL, CURRENT PACKAGE PATH,
VOLATILE Table Support, Star Join Sparse Index, Qualified Column names,
Multiple DISTINCT clauses, ON COMMIT DROP, Transparent ROWID Column,
call from trigger, statement isolcation, FOR READ ONLY KEEP UPDATE LOCKS
SET CURRENT SCHEMA, client special registers, long SQL Object names,
SELECT FROM INSERT, UPDATE, DELETE, MERGE, INSTEAD OF TRIGGER,
Native SQL Procedure Language, BIGINT, file reference variables, XML, FETCH
FIRST & ORDER BY IN subselect and fullselect, caseless comparisons,
INTERSECT, EXCEPT, not logged tables, XXXXXX, XXXXX XXXX XXXXXXXX,
XXXXXX, XXXX,XXXX,XXXXXXX XXXXX, XXXXXXX XXXXX XXXXXX,
XXXX,XXXXXX XXXXXX, XXXXX XXXX, XXXXXX,XXXXXX,XXXXXX, XXXXXX
XXXXXX XXXXX XXXXX, XXXXXXXXX XXXXXXXXXXX
XXXXXXX XXXXX, XXXXXX, XXXX, XX XXXXXXX XXXXXXXX
© Sheryl M. Larsen, Inc. 2006-2007
8

Notes:

Characteristics of SQL Gone WILD

Spans all industries

Notes:

Characteristics of SQL Gone Wild

- Millions of GETPAGES
- Millions of SELECT INTOs
- High wait for I/O count
- High elapsed time
- High CPUPCT for OPEN
- SQLCODE -101
- Misuse of SQL
- Stage 2 conditions

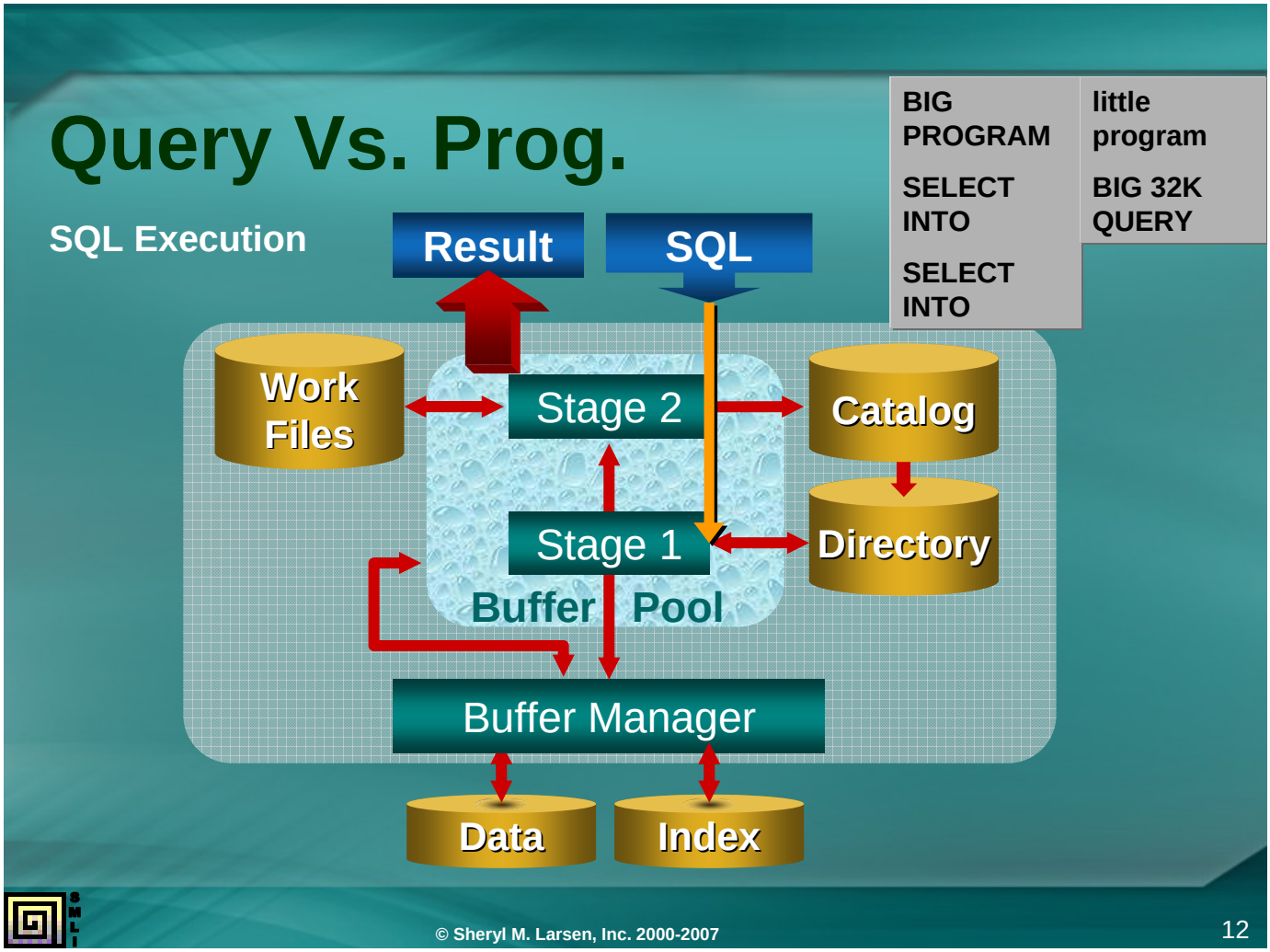
```
SSID ==> DB22           Planname ==> MYPLAN        Program ==> PROG384
SQL_CALL  STMT#  SECT#  SQL        TIMEPCT CPUPCT  INDB2_TIME
--------  -----  -----  ---------- ------- ------- ------------
 _ OPEN     01367  00001         361  99.80%  97.98% 00:15.542845
 _ FETCH    01394  00001         361    .12%   1.29% 00:00.006963
 _ CLOSE    01559  00001         361    .05%    .57% 00:00.003126
 _ FETCH    01478  00001          27    .01%    .14% 00:00.000788
```


Notes:

High Getpage Count

DB2 Dur	DB2 CPU	Appl Dur	Applictn CPU	Getpage
30:01	20:58.244	30:01	20:58.247	**169550K**
10:14	6:41.339	10:14	6:41.341	37812K
38:13	9:45.571	38:26	9:46.828	34587K
26:48	5:22.940	27:00	5:30.694	26000K
9:14	6:24.965	9:14	6:25.029	23628K
12:14	5:38.048	12:27	5:38.702	23153K
11:39	3:16.830	11:40	3:16.945	22740K
15:55	9:26.651	16:31	9:32.636	22334K
48:21	9:55.964	48:37	9:59.777	21797K
24:10	4:32.137	24:19	4:38.782	21255K
3:17	2:23.844	5:38	2:40.243	19548K
24:10	4:04.909	24:25	4:11.109	18979K
13:16	5:55.802	13:16	5:55.951	18169K

Notes:



Notes:

Stage 1 has grown up over the past 20 years and can now process 30 more WHERE clauses than it could in 1983 for a total of 33 (21 indexable, 12 nonindexable), 36 as of V7 and 44 as of V8

This has significantly impacted how high performing SQL applications should be coded.

Just because this functionality/technology is in place does not mean that it is always used to its fullest potential.

It is optional for a developer to use it or not. IBM **wins** either way.

Developer's companies **wins** when the technology is exploited through increase throughput and quicker response times.

SQLCODE -101

- 32K Query

```
,F.LOCAL_INTBS_TRD_DT
,F.LOCAL_INV_COST
,CASE
WHEN F.SUB_TXN_CODE = 'IS'
 OR F.SUB_TXN_CODE = 'ISC' THEN 0
ELSE F.LOCAL_INV_G_L END AS LOCAL_INV_G_L
,CASE
WHEN F.SUB_TXN_CODE = 'IS'
 OR F.SUB_TXN_CODE = 'ISC' THEN 0
ELSE F.LOCAL_INV_G_L_SS END AS LOCAL_INV_G_L_SS
,CASE
WHEN F.SUB_TXN_CODE = 'IS'
 OR F.SUB_TXN_CODE = 'ISC' THEN 0
ELSE F.LOCAL_INV_G_L_ST END AS LOCAL_INV_G_L_ST
,F.LOCAL_PRICE
,CASE
 WHEN (F.SUB_TXN_CODE  = 'FX' OR F.SUB_TXN_CODE   = 'FXC'
 OR    F.CASH_CATEGORY = '035' OR F.CASH_CATEGORY = '060')
 AND  (F.FX_PAYABLE_CCY = K.CURC_CODE)
    THEN F.UNITS_1
 WHEN
    (F.SUB_TXN_CODE  = 'FX' OR F.SUB_TXN_CODE   = 'FXC'
OR   F.CASH_CATEGORY = '035' OR F.CASH_CATEGORY = '060')
AND (F.FX_PAYABLE_CCY <> K.CURC_CODE)
    THEN F.UNITS_2
      WHEN F.SUB_TXN_CODE = 'CS'  OR
           F.SUB_TXN_CODE = 'CSC' OR
```


Notes:

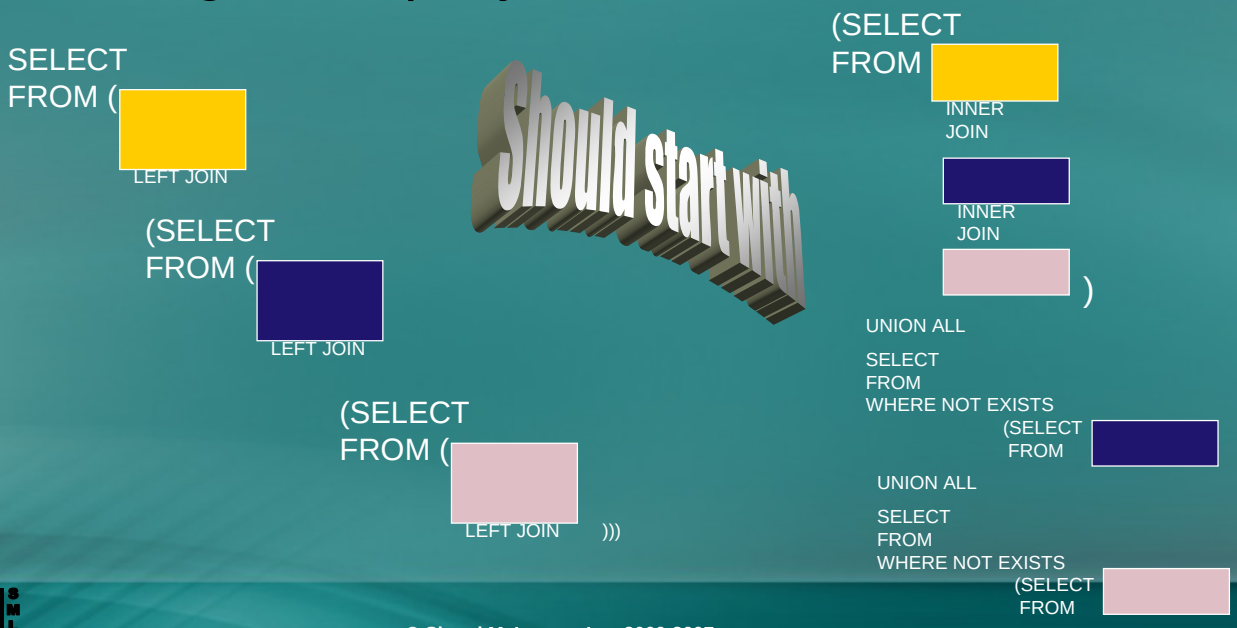
Getting More Complex on Z

```
UNION
    SELECT 'S' ,
        F . FK_SUSPENSE_CODE ,
        E . QUOTE_TYPE_CODE ,
        COUNT ( * ) ,
        SUM ( CASE WHEN B . DELIVERY_PLAN_DT =
        :  WS-ETA-DATE THEN 1 ELSE 0 END ) ,
        SUM ( E . HANDLING_UNIT_CNT ) ,
        SUM ( E . TOTAL_PIECES_QTY ) ,
        SUM ( E . TOTAL_WEIGHT_QTY )
    FROM   SHIPMENT_ROUTE A ,
        SUSPENSE_LOAD_TO F ,
        SHIPMENT_LOAD_TO B ,
        SHIPMENT_LOCATION C ,
        EQUIPMENT D ,
        SHIPMENT E
    WHERE  A . DEST_CC_ID =
        :  WS-TERMID
    AND    ( A . DEST_ARRIVAL_DATE <
        :  WS-ETA-DATE
    OR     ( A . DEST_ARRIVAL_DATE =
        :  WS-ETA-DATE
    AND    A . DEST_ARRIVAL_TIME <=
        :  WS-ETA-TIME ) )
    AND    A . GROUP_DEST_CC_ID = ' '
    AND    F . FK_PRO_NUMBER = A . PRO_NUMBER
    AND    F . FK_SUFFIX_NUMBER = A . SUFFIX_NUMBER
```

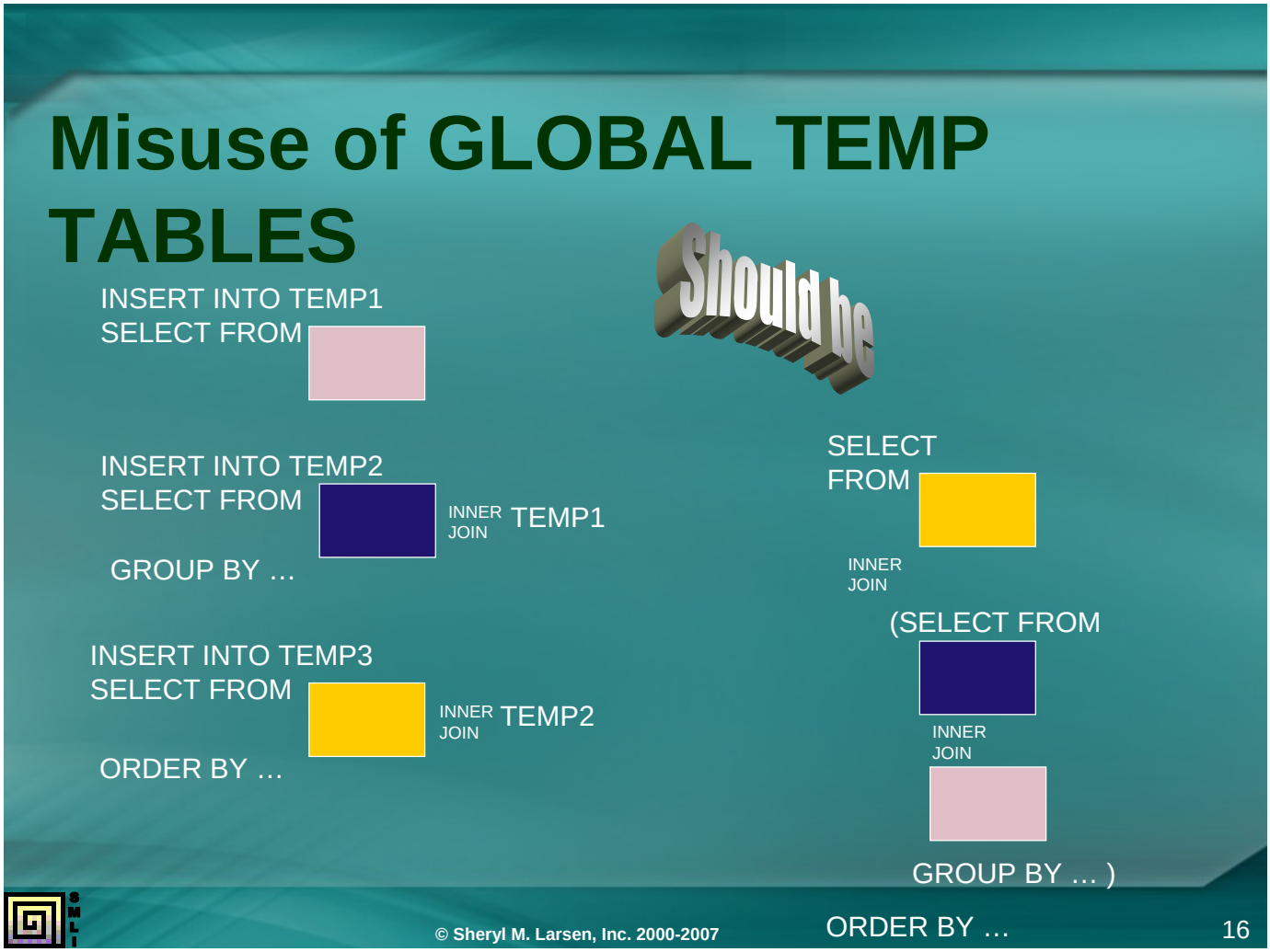

Notes:

Over Use of LEFT JOIN

- Paying to get exceptions and dragging them through multiple joins



Notes: LEFT/RIGHT JOINs are usually more expensive that INNER JOINs on the same tables due to the fact that the answer set includes exceptions, rows in one table that do not hook up to the other table. If those exceptions contain critical information, then by all means, use LEFT/RIGHT JOINs. In many reporting applications, however, the usefulness of the exceptions may not be obvious. Consider leaving them outside the main INNER JOIN query block and segregating the data, using NOT EXISTS, on the report for approval of the exception data. Once they are justified, convert to LEFT/JOINs.



Notes:

The most common misuse is in multi-step processing. The example above shows the use of three GLOBAL TEMPs in a three step process. These GLOBAL TEMP only hold the data for one step and there is zero intermediate processing occurring between steps. This easily converts to one query with multiple table expressions separating the steps when necessary. The performance improvement is significant due to the elimination of creation, load and retrieval of three TEMP tables.

Sources of SQL Gone WILD

With Examples

Notes:

Sources of SQL Gone Wild

- Developers with low SQL skill level
- Code G E N E R A T O R S with low SQL skill level
- Where do they go wrong?

SQL Skill Self Assessment

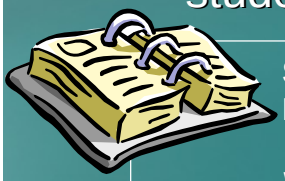
Level	Assessment
0	Think SQL is an energy drink
1	Simple SELECT statements, WITH clause, EXPLAIN, ORDER BY
2	WHERE clauses, BETWEEN, LIKE, IN(list), =, >=, >, <, <=, <>, NOT IN(list), NOT LIKE, NOT BETWEEN
3	Table joins (inner, outer, full), UNION, UNION ALL, SQLCA, CONCAT, static CURSORs, FOR UPDATE OF, UOW/RID
4	noncorrelated and correlated subqueries, EXISTS, NOT EXISTS, FETCH FIRST x ROWS ONLY, OPTIMIZE FOR x ROWS
5	Indexable, Stage1 and Stage 2 predicate evaluation, multirow FETCH/INSERT, GET DIAGNOSTICS, Scalar full SELECT
6	Table expressions/common table expressions, GROUP BY, HAVING, IS NOT DISTINCT FROM, embedded dynamic SQL/SQLDA
7	CASE expressions, GTT, DTT, Dynamic Scrollable cursors, SEQUENCES, GROUP BY EXPRESSION
8	Queries involving > 10 tables, INSERT within SELECT, Star Schema, Snow Flake
9	MQT (Materialized Query Tables), Recursive SQL, UNION in Views, Triggers, Stored Procedures, > 20 SQL Functions
10	Codes effective and efficient SQL and knows when to use each appropriately

18

Notes:

Joins over Subqueries

- Needed for Intra-Table Column Comparisons
  - » WHERE A.COL2 = A.COL5
  - » On different rows (accounts, policies, items, orders, other unique ids, etc.)
  - » What Students Have the Same Classes as any given student?



```
SELECT  SSS.SID, SSS.CLASS_ID, SSS.TIME_ID
FROM    SMLU_STUDENT_SCHED SS
       ,SMLU_STUDENT_SCHED SSS
WHERE  SSS.CLASS_ID = SS.CLASS_ID
  AND  SSS.TIME_ID = SS.TIME_ID
  AND  SSS.SID <> SS.SID
  AND  SSS.SID NOT IN(list challenged student IDs)
  AND  SS.SID = :challenged-sid
```

Self Join to get same class/time rows

Point to Different Rows

Homework Challenged

19

Notes:

Subqueries over Joins

- Needed for Intra-Table Column Comparisons
  - » What Students Have **at least one** of the same Classes as a given student?

```
SELECT  SS.SID
FROM    SMLU_STUDENT_SCHED SS
WHERE  SS.SID NOT IN(list challenged student IDs)
 AND  EXISTS
          (SELECT 'TRUE OR FALSE'
          FROM SMLU_STUDENT_SCHED SSS
          WHERE  SSS.CLASS_ID = SS.CLASS_ID
             AND   SSS.TIME_ID = SS.TIME_ID
             AND   SSS.SID <> SS.SID
             AND    SSS.SID = :challenged-sid)
```

Homework Challenged

Notes:

Solutions for Taming

Rewrite and Tuning

Notes:

Solutions for Taming SQL

Manual Query re-write

Value BETWEEN COL1 AND COL2 is Stage2

Rewrite:

(Value >= COL1 AND value <= COL2

WHERE DATE(col_TS) >= :date1

Rewrite:

WHERE col_TS >=

TIMESTAMP(:date1, '00:00:000')

Notes:

Solutions .....

- Manual GROUP BY pushdown

```
SELECT C.CUST_ID
                ,MIN(C.CUST_NAME) AS CUST_NAME
                ,MIN(C.CUST_PHONE)AS CUST_PHONE
                ,SUM(S.SALES) AS TOTAL_SALES
FROM            CUSTOMER C, SALES  S
WHERE           C.CUST_ID = S.CUST_ID
    AND         S.SALES_DATE BETWEEN :date-lo AND :date-hi
GROUP BY        C.CUST_ID
```


```
SELECT C.CUST_NAME, C.CUST_PHONE, S.TOTAL_SALES
FROM    CUSTOMER C
 ,      (SELECT         S.CUST_ID, SUM(S.SALES) AS TOTAL_SALES
         FROM           SALES  S
         WHERE          S.SALES_DATE BETWEEN :date-lo AND :date-hi
         GROUP BY       S.CUST_ID)  AS  S
WHERE C.CUST_ID = S.CUST_ID
```


Notes:

How to Tune Queries

- Sheryl's Tuning Tools
  - » OPTIMIZE FOR n ROWS
  - » FETCH FIRST n ROWS ONLY
  - » No Op (+0, CONCAT ' ')
  - » ON 1=1
  - » Fake Filters
    - COL BETWEEN :hv1 AND :hv2
    - COL >= :hv
    - TX.COL = TX.COL
  - » Table expressions with DISTINCT
    - FROM (SELECT DISTINCT COL1, COL2 .....)
  - » REOPT(VARS)
  - » Extreme Cross Queryblock Optimization (X2QBOpt)
  - » Index and MQT Optimization

Notes:

Manual Query Rewrite

For Extreme Cases

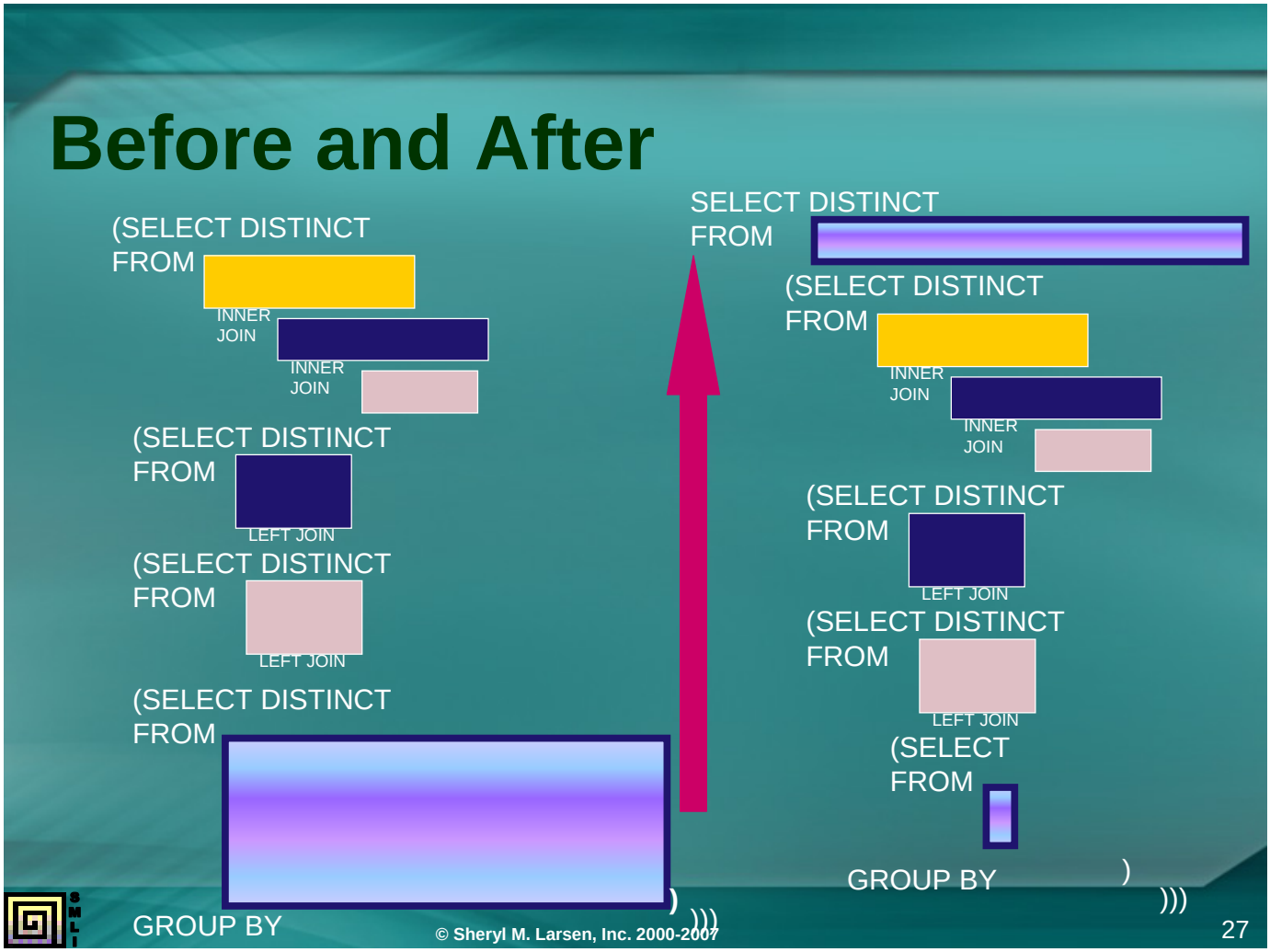
Notes:

A Typical Data Warehouse Query

- Initial cost of 16 million timerons
  - » WOULD NOT FINISH!
- Multiple DISTINCT Table Expressions
- Initial join involved all columns and all rows
- The very wide and very deep set was dragged through many more query steps

Notes:

As queries get more complex, intra query optimization becomes necessary. Cross queryblock knowledge can greatly assist the optimizer in query rewrite. Right now this is a manual rewrite process. One example is a statement that contained multiple UNION ALL subselects with the initial subselect involving and join requesting most rows all columns. This very wide and very deep set was dragged through many query steps.



Notes:

The outermost query block, the last step, requested a GROUP BY ROLLUP. This query would not even finish and the timeron value was over 16 million. To manually rewrite this query, the largest block was analyzed for the columns required for GROUP BY, and remaining LEFT JOINs. The initial SELECT list for that really wide table expression was then manually pruned down to SELECT only critical columns, but all rows. This essentially put the subselect on a diet so that the next 5 join steps were much narrower. The final step was then rewritten to join back to the tables to get the remaining SELECT list columns. This did increase the number of times that main set was accessed but the savings from the wide joins more than offset the cost.

Further analysis was done on the ROLLUP columns. It was determined that the only columns needed in the ROLLUP calculation were from the main set. The ROLLUP operation was moved from the outer most step and pushed into the first table expression. This greatly reduced the cost of the GROUP BY operation since it did not involve many columns.

Tuning Technique – X2QBOpt

- Extreme Cross Query Block Optimization
- Identify and pre-qualify the core set of data and only select the keys early on
- Once all the steps are complete, go back and get the remaining columns
- Referred to in class as “Separating the GROUP BY Work”
  - » Keeping it thin through the DB2 engine
- Brought cost down to 270,000 timerons
  - » Query now finishes!

Notes:

The query now finishes and the timerons were reduced to .27 million. This technique of keeping the query thin through the DB2 engine has to be accomplished through manual query rewrite for now. Start by identifying the core set of data and only select the keys and grouping columns early on. Once all the step are complete, go back and get the remaining columns.

Solutions for Prevention

Best Practice Shops

Notes:

Solutions for Prevention

- SQL
  - » Query development prior to program development
  - » SQL skill level 7-10
  - » Continual SQL education
- Code reuse
  - » Encapsulation of common processes into Stored Procedures
  - » Popular processing pushed into SQL using User Defined Functions, UDFs
- IBM Certification – a good start
- Internal Certification Test
  - » Includes SQL, Data Model, Life Cycle Policies/procedures
  - » Used to rank staff to optimize project staffing

Notes:

Future Solutions for Prevention

- IBM DB2 Query Rewrite
  - » Adds cost to every query optimized
- Tools for Query Rewrite
  - » Adds cost only to queries selected
- IBM DB2 Query Optimization
  - » Adds time to every query for that level of optimization
- Tools for Query Optimization
  - » IBM DB2 Optimization Expert for z/OS will assist in analysis and recommendations for query tuning in DB2 9 for z/OS – became generally available March 16, 2007
  - » http://www-306.ibm.com/software/data/db2/zos-new/
- Access Path Stabilization
  - » Freezing the access path through migrations

Notes:

Performance Recommendations

- ✓Physical environment impacts workload performance
- ✓System
  - ✓ Bufferpool separation for the workload pattern
  - ✓ Schema design for the workload pattern
  - ✓ Use of Constraints to assist the optimizer
- ✓Application
  - ✓ **SQL exploitation and proper use**
  - ✓ **Manual Query Rewrite when necessary**
  - ✓ Optimal Index and MQT design for the workload pattern

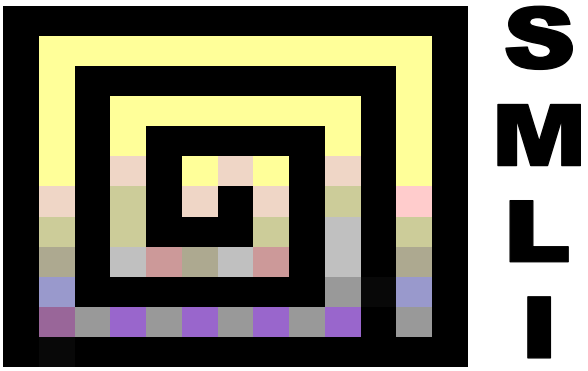
Notes:

A well thought out physical environment will yield a positive impact on performance. System factors such as bufferpool and schema should be aligned with the workload pattern. Constraints, even just informational, greatly assist today's optimizer. Application factors such as smart SQL and manual query rewrite for the workload pattern can impact performance up to 80%. Optimal index and MQT design are the second line of defense for performance. There are many strategies but the most optimal revolves around the discovery of patterns of SQL.

SQL Gone WILD!

Session: G04 or G12
Solutions for SQL Gone Wild

Sheryl M. Larsen
Sheryl M. Larsen, Inc.
SherylMLarsen@cs.com



GoFurther

Notes: