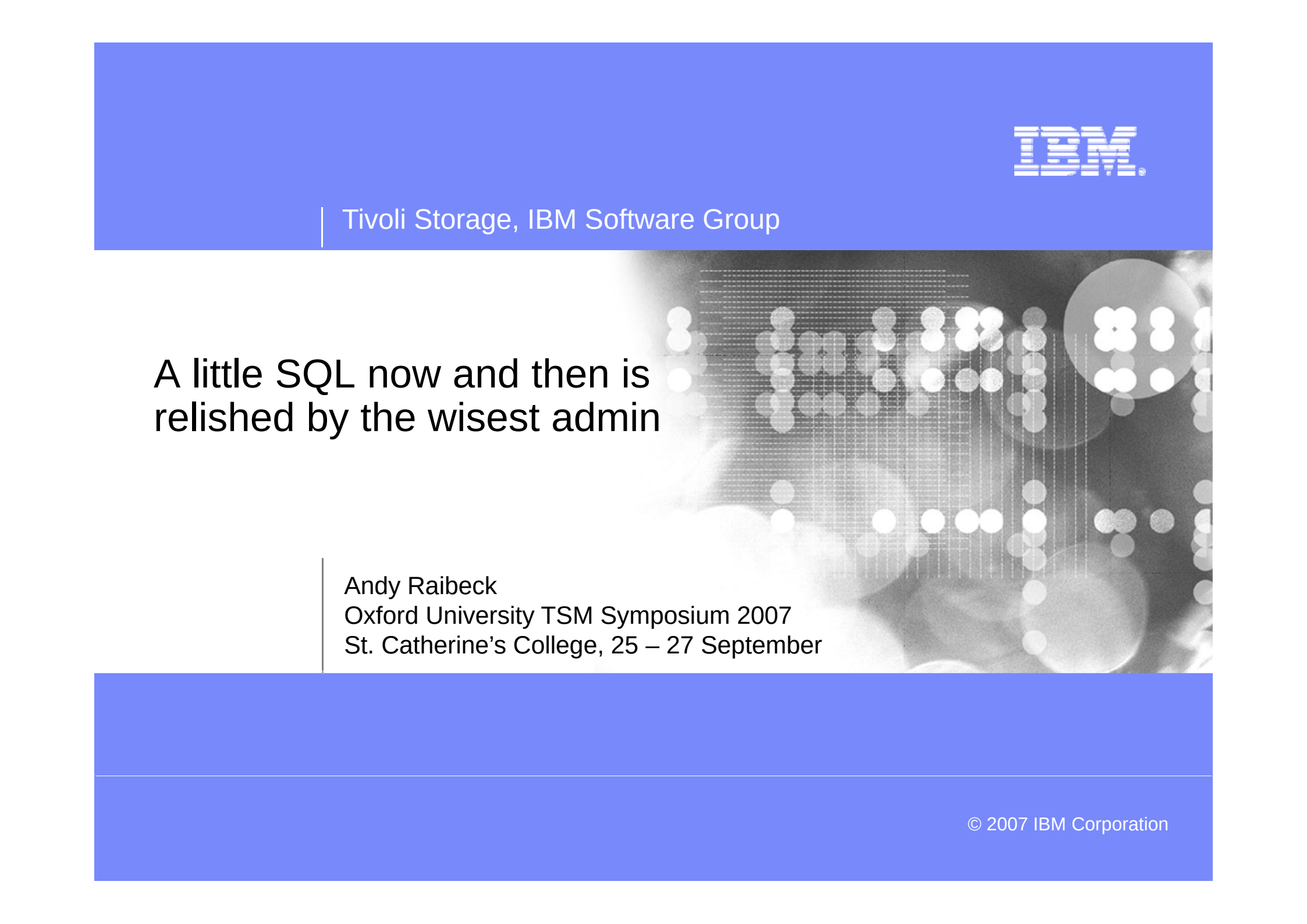

IBM
Tivoli Storage, IBM Software Group
A little SQL now and then is relished by the wisest admin
Andy Raibeck
Oxford University TSM Symposium 2007
St. Catherine's College, 25 – 27 September
© 2007 IBM Corporation

Trademarks

- The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
  - AIX
  - IBM
  - Tivoli
- Other company, product, and service names may be trademarks or service marks of others.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.

Abstract

One of the most frequently asked class of questions about Tivoli Storage Manager is how to query the TSM server database in order to generate custom reports. This session will focus on TSM's built-in support for the SQL SELECT statement: SELECT tutorial; Sample SELECT statements; Q & A.

Agenda

- Getting started
- An SQL primer
- Using SQL to query the TSM server
- Other considerations

Getting started

- How do I find out what kind of information I can query?
- Three system catalog tables:
  - SYSCAT.TABLES
  - SYSCAT.COLUMNS
  - SYSCAT.ENUMTYPES
- A good place for an introduction to SQL.

The simplest SELECT statement

- This is the simplest form of the SELECT statement:

  `SELECT * FROM tablename`

- FROM specifes the table to query.
- In this example, all table rows are returned.
- The SELECT part specifies which columns.

  * means all columns.

Table names

- Table names have this format:

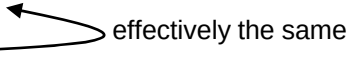
  schemaname.*tablename*

- The TSM database has two schemae: ADSM and SYSCAT.
- The "*schemaname*." part may be omitted if there is no ambiguity in the table name.
- There are no same-name tables between the ADSM and SYSCAT schemae.

Table names – simple examples

- Examples:
  - Which tables can I query?

```
SELECT * FROM SYSCAT.TABLES
SELECT * FROM TABLES
```



  - What columns are in the tables?

```
SELECT * FROM COLUMNS
```

  - What enumerated types does the TSM server have?

```
SELECT * FROM ENUMTYPES
```

The TABLES table

```
tsm: SS2_ADSM_GROUP_SERVER>select * from tables

     TABSCHEMA: ADSM
       TABNAME: ACTLOG
   CREATE_TIME: 1998-02-10 13:29:21.000001
      COLCOUNT: 13
INDEX_COLCOUNT: 1
  UNIQUE_INDEX: FALSE
       REMARKS: Server activity log

     TABSCHEMA: ADSM
       TABNAME: ADMINS
   CREATE_TIME: 1998-02-10 13:29:21.000001
      COLCOUNT: 18
INDEX_COLCOUNT: 1
  UNIQUE_INDEX: TRUE
       REMARKS: Server administrators

     TABSCHEMA: ADSM
       TABNAME: ADMIN_SCHEDULES
   CREATE_TIME: 1998-02-10 13:29:21.000001
      COLCOUNT: 20
INDEX_COLCOUNT: 1
  UNIQUE_INDEX: TRUE
       REMARKS: Administrative command schedules
...
```

The TABLES table (*continued*)

- The columns of interest from the TABLES table are:
  - **TABNAME:** Name of the table.
  - **COLCOUNT:** Number of columns in the table.
  - **INDEX_COLCOUNT:** Number of indexed columns in the table.
  - **REMARKS:** Brief description of the table.

Customizing which columns are shown

- How do I display only the columns I am interested in?

  `SELECT` *col*$_1$`[,`*col*$_2$`,...,`*col*$_n$`]` `FROM` *tablename*

- This displays only the requested columns from the table specified by *tablename*.
- Columns will be displayed in the specified order.

Customizing which columns are shown - Examples

- Examples:

```
SELECT TABNAME FROM TABLES

SELECT TABNAME,REMARKS FROM TABLES

SELECT REMARKS,TABNAME,INDEX_COLCOUNT FROM TABLES
```

Customizing which columns are shown – Examples (*continued*)

```
tsm: SS2_ADSM_GROUP_SERVER>SELECT REMARKS,TABNAME,INDEX_COLCOUNT FROM TABLES

REMARKS                 TABNAME                 INDEX_COLCOUNT
------------------      ------------------      --------------
Server activity         ACTLOG                               1
 log
Server administra-      ADMINS                               1
 tors
Administrative          ADMIN_SCHEDULES                      1
 command schedules
Table Name              AFBF                                 3
Table Name              AFSG                                 4
Client archive          ARCHIVES                             4
 files
Management class        AR_COPYGROUPS                        4
 archive copy
...
```

Customizing column headings

- You can change the column headings on the output to give them more meaning or make them more readable:

```
SELECT col1 [AS] "heading1",col2 [AS] "heading2",...,coln [AS] "headingn"
  FROM tablename
```

- Examples:

```
SELECT TABNAME AS "Table Name",REMARKS FROM TABLES

SELECT REMARKS AS "Remarks",TABNAME as "Table Name",
       INDEX_COLCOUNT AS "Num. Indices" FROM TABLES
```

Customizing column headings (*continued*)

```
tsm: SS2_ADSM_GROUP_SERVER>SELECT REMARKS AS "Remarks",TABNAME as "Table Name",INDEX_COLCOUNT AS "Nu
m. Indices" FROM TABLES

Remarks              Table Name             Num. Indices
------------------   ------------------     ------------
Server activity      ACTLOG                            1
 log
Server administra-   ADMINS                            1
 tors
Administrative       ADMIN_SCHEDULES                   1
 command schedules
Table Name           AFBF                              3
Table Name           AFSG                              4
Client archive       ARCHIVES                          4
 files
Management class     AR_COPYGROUPS                     4
 archive copy
 groups
Client schedule      ASSOCIATIONS                      2
 associations
```

Row filtering – the WHERE clause

- So far we've performed operations that retrieve all rows from a table.
- To retrieve only certain rows, use the WHERE clause:

  `SELECT` *`columnlist`* `FROM` *`tablename`* `WHERE` *`predicate`*

- *predicate* evaluates to a condition that causes only certain rows to be displayed.

WHERE clause predicates

- Sample predicates:

  col_1=*value*

  NOT col_1>*value*

  col_1>*value* AND col_2<*value*

  col_1=*value* OR col_2>=*value*

  NOT ((col_1=*value* OR col_2<*value*) AND col_3=*value*)

WHERE clause predicates (*continued*)

- Sample predicates (*continued*):

  `col1 LIKE 'pattern' [ESCAPE character]`

  – Pattern matching:
  - `_` (underscore) matches exactly one character
  - `%` (percent) matches zero or more characters
  - Any other character matches itself
  - Use `ESCAPE` followed by *character* if you need to use the actual '_' or '`%`' characters in your pattern
    – *character* becomes an escape character you can use in your pattern to prevent '_' or '`%`' from being used for pattern matching

WHERE clause predicates (*continued*)

- Pattern matching examples:

`'ANR9999%'`:
Any string beginning with 9999 followed by 0 or more characters

`'VOL00_'`
Any string beginning with VOL00 followed by exactly one character

`'%_.exe'`
Any string with at least one character followed by .exe

`'%STORMAN\_%' ESCAPE '\'`
Any string containing STORMAN_

WHERE clause predicates (*continued*)

```
adsm> select tabname as "Table", colname as "Column" from columns where colname like
      '%\_NAME' escape '\'

Table                   Column
--------------------    --------------------
ADMINS                  ADMIN_NAME
ADMIN_SCHEDULES         SCHEDULE_NAME
ARCHIVES                NODE_NAME
ARCHIVES                FILESPACE_NAME
ARCHIVES                HL_NAME
ARCHIVES                LL_NAME
ARCHIVES                CLASS_NAME
AR_COPYGROUPS           DOMAIN_NAME
AR_COPYGROUPS           SET_NAME
AR_COPYGROUPS           CLASS_NAME
AR_COPYGROUPS           COPYGROUP_NAME
ASSOCIATIONS            DOMAIN_NAME
ASSOCIATIONS            SCHEDULE_NAME
ASSOCIATIONS            NODE_NAME
AUDITOCC                NODE_NAME
BACKUPS                 NODE_NAME
BACKUPS                 FILESPACE_NAME
BACKUPS                 HL_NAME
BACKUPS                 LL_NAME
BACKUPS                 CLASS_NAME
BACKUPSETS              NODE_NAME
BACKUPSETS              BACKUPSET_NAME
...
```

WHERE clause predicates (*continued*)

- Some other predicates:

  col_1 BETWEEN $value_1$ AND $value_2$
  - BETWEEN is inclusive

  col_1 NOT BETWEEN $value_1$ AND $value_2$
  - NOT BETWEEN is exclusive

  col_1 IN ($value_1$, $value_2$, ..., $value_n$)

  col_1 NOT IN ($value_1$, $value_2$, ..., $value_n$)

  col_1 IS NULL

  col_1 IS NOT NULL

WHERE clause predicates - Examples

- Examples:

```
select node_name, type, filespace_name,
       num_files, logical_mb
   from occupancy
   where num_files between 1 and 50


select node_name, filespace_name,
       num_files, logical_mb
   from occupancy
   where num_files between 0 and 500000 and
         type='Bkup' and
         node_name in ('STORMAN','RAIBECK')
```

WHERE clause predicates – Examples (*continued*)

```
tsm: SS2_ADSM_GROUP_SERVER>select node_name, filespace_name, num_files,
logical_mb from occupancy where num_files between 0 and 500000 and type='Bkup' and
node_name in ('STORMAN','RAIBECK')

ANR2963W This SQL query may produce a very large result table, or may require a
significant amount of time to compute.

Do you wish to proceed? (Yes (Y)/No (N)) y

NODE_NAME            FILESPACE_NAME          NUM_FILES          LOGICAL_MB
-------------------  ------------------    -----------     ----------------
RAIBECK              \\amr-laptop\c$_o-          55920            14146.35
                      ld
RAIBECK              \\amr-laptop\c$_o-          55920            14146.35
                      ld
RAIBECK              \\amr-laptop\c$_o-          33795            11386.27
                      ld2
RAIBECK              \\amr-laptop\c$_o-          33795            11386.27
                      ld2
...
STORMAN              SYSTEM OBJECT                2732              549.75
STORMAN              SYSTEM OBJECT                2732              549.75
STORMAN              ASR                             7                0.04
STORMAN              ASR                             7                0.04
```

Aggregate functions

- Aggregate functions perform operations on values from selected rows to produce a single value.
- Aggregate functions include `COUNT`, `SUM`, `AVG`, `MAX`, and `MIN`.
- `COUNT` is useful for finding the number of rows that match a query.

Aggregate functions (*continued*)

- How many scratch volumes do I have in my library?

```
tsm: SS2_ADSM_GROUP_SERVER>select count(*) as "# Scratch Vols"
from libvolumes where status='Scratch'

# Scratch Vols
--------------
            12

tsm: SS2_ADSM_GROUP_SERVER>
```

- TSM eccentricity: not all string literal values are upper case.
  - When in doubt, you can use function `LCASE()` or `UCASE()`.

```
... where lcase(status)='scratch'
... where ucase(status)='SCRATCH'
```

Math 101

- You can do math with the SELECT column specifications:

```
tsm: SS2_ADSM_GROUP_SERVER>select stgpool_name, maxscratch,
100 * numscratchused / maxscratch as "% USED",
maxscratch - numscratchused as "REMAINING SCRATCH" from stgpools

STGPOOL_NAME             MAXSCRATCH          % USED     REMAINING SCRATCH
------------------     -----------     -----------     -----------------
ARCHIVEPOOL
BACKUPPOOL
DISKPOOL
FILEPOOL                        200              90                  19
HSMPOOL                         200               0                 200
LTOCOPY                          75              52                  36
LTOPOOL                          75              50                  37

tsm: SS2_ADSM_GROUP_SERVER>
```

- What is wrong with this picture?

Math 101 (*continued*)

- Filter out non-sequential pools and switch from integer to floating-point math

```
tsm: SS2_ADSM_GROUP_SERVER>select stgpool_name, maxscratch,
100.0 * numscratchused / maxscratch as "% USED", maxscratch - numscratchused as "REMAINING SCRATCH"
from stgpools where devclass != 'DISK'

     STGPOOL_NAME: FILEPOOL
       MAXSCRATCH: 200
           % USED: 90.500000000000000000
REMAINING SCRATCH: 19

     STGPOOL_NAME: HSMPOOL
       MAXSCRATCH: 200
           % USED: 0.000000000000000000
REMAINING SCRATCH: 200

     STGPOOL_NAME: LTOCOPY
       MAXSCRATCH: 75
           % USED: 52.000000000000000000
REMAINING SCRATCH: 36

     STGPOOL_NAME: LTOPOOL
       MAXSCRATCH: 75
           % USED: 50.666666666666666666
REMAINING SCRATCH: 37


tsm: SS2_ADSM_GROUP_SERVER>
```

- Now what?

Math 101 (*continued*)

- Remove excess fractional digits

```
tsm: SS2_ADSM_GROUP_SERVER>select stgpool_name, maxscratch,
cast(100.0 * numscratchused / maxscratch as dec(5,2)) as "% USED",
maxscratch - numscratchused as "REMAINING SCRATCH" from stgpools
where devclass != 'DISK'

STGPOOL_NAME             MAXSCRATCH      % USED     REMAINING SCRATCH
------------------     -----------     -------     -----------------
FILEPOOL                       200       90.50                    19
HSMPOOL                        200        0.00                   200
LTOCOPY                         75       52.00                    36
LTOPOOL                         75       50.66                    37

tsm: SS2_ADSM_GROUP_SERVER>
```

- Rounding?

Math 101 (*continued*)

- 50.6666… should round to 50.67
- By default TSM truncates digits in the cast
- We can achieve rounding to two decimal places by adding 0.005 to the value before the cast
  - 50.66666… + 0.005 = 50.67166…
  - After the cast and truncation, we will be left with 50.67

Math 101 (*continued*)

- Rounding the hard way…

```
tsm: SS2_ADSM_GROUP_SERVER>select stgpool_name, maxscratch,
cast(100.0 * numscratchused / maxscratch + 0.005 as dec(5,2))
as "% USED", maxscratch - numscratchused as "REMAINING SCRATCH"
from stgpools where devclass != 'DISK'

STGPOOL_NAME            MAXSCRATCH      % USED     REMAINING SCRATCH
------------------     -----------     -------     -----------------
FILEPOOL                       200      90.50                    19
HSMPOOL                        200       0.00                   200
LTOCOPY                         75      52.00                    36
LTOPOOL                         75      50.67                    37

tsm: SS2_ADSM_GROUP_SERVER>
```

Math 101 (*continued*)

- Rounding the easier way…

```
tsm: SS2_ADSM_GROUP_SERVER>set sqlmathmode round

tsm: SS2_ADSM_GROUP_SERVER>select stgpool_name, maxscratch, cast(100.0 * numscra
tchused / maxscratch as dec(5,2)) as "% USED", maxscratch - numscratchused as "R
EMAINING SCRATCH" from stgpools where devclass != 'DISK'

STGPOOL_NAME            MAXSCRATCH      % USED     REMAINING SCRATCH
------------------     -----------     -------     -----------------
FILEPOOL                       200       92.00                    16
HSMPOOL                        200        0.00                   200
LTOCOPY                         75       52.00                    36
LTOPOOL                         75       50.67                    37

tsm: SS2_ADSM_GROUP_SERVER>
```

- Much better!
- Nit: add to `where` clause `and maxscratch <> 0`
- Alternative: `where maxscratch is not null and maxscratch >0`

OCCUPANCY table example

- Another example: How much backup storage is STORMAN using?

```
tsm: SS2_ADSM_GROUP_SERVER>select * from occupancy
where node_name = 'STORMAN'

     NODE_NAME: STORMAN
          TYPE: Bkup
FILESPACE_NAME: \\storman\c$_2006-05-10
  STGPOOL_NAME: LTOCOPY
     NUM_FILES: 138183
   PHYSICAL_MB: 16878.65
    LOGICAL_MB: 16878.65
  FILESPACE_ID: 1

     NODE_NAME: STORMAN
          TYPE: Bkup
FILESPACE_NAME: \\storman\c$_2006-05-10
  STGPOOL_NAME: LTOPOOL
     NUM_FILES: 138183
   PHYSICAL_MB: 16878.65
    LOGICAL_MB: 16878.65
  FILESPACE_ID: 1
...
```

OCCUPANCY table example (*continued*)

- This is a good way to see what's available in a table
  - We only want backup, so we'll need to filter on TYPE
  - We are not concerned with file space name, storage pool name, or file space ID.
  - So let's take the next step…

OCCUPANCY table example (*continued*)

- Another example: How much backup storage is STORMAN using?

```
tsm: SS2_ADSM_GROUP_SERVER>select num_files,
physical_mb, logical_mb from occupancy
where node_name = 'STORMAN' and type='Bkup'

 NUM_FILES          PHYSICAL_MB          LOGICAL_MB
-----------     ----------------     ----------------
     138183             16878.65             16878.65
     138183             16878.65             16878.65
     186807             24962.25             24962.25
...
```

- This looks like the kind of data we're after, so let's add it up.

OCCUPANCY table example (*continued*)

- This uses the `SUM` aggregate function:

```
tsm: SS2_ADSM_GROUP_SERVER>select sum(num_files) as "Num. files",
sum(physical_mb) as "Physical MB", sum(logical_mb) as "Logical MB"
from occupancy where node_name = 'STORMAN' and type = 'Bkup'

 Num. files                      Physical MB                      Logical MB
----------- --------------------------------- ---------------------------------
    5258006                         539721.73                         531174.60
```

- You can cast the 2nd and 3rd columns as, say, decimal(17,2) to reduce the width, e.g.:

```
cast(sum(physical_mb) as decimal(17,2)) as "Physical MB"
```

OCCUPANCY table example – GROUP BY

- Now let's say we want this same occupancy information, but for all nodes
- `GROUP BY` is used to define groups of rows to which aggregate functions can be applied
- Each item in the `SELECT` list must produce a single value per group

OCCUPANCY table example – GROUP BY (*continued*)

- For this example, we will group occupancy information by node

```
tsm: SS2_ADSM_GROUP_SERVER>select node_name, sum(num_files) as "Num. files", sum(physical_mb)
as "Physical MB", sum(logical_mb) as "Logical MB" from occupancy where type = 'Bkup' group by node_name

NODE_NAME           Num. files                      Physical MB                      Logical MB
------------------ ----------- --------------------------------- ---------------------------------
3A6929                  283624                          50761.14                         50721.72
ADSMAPAR                  9368                            235.74                           235.74
ADSMPERF                 11628                            269.24                           269.24
AKOJENOV                238948                          27951.52                         27814.88
ALAESTANTE              121040                           9698.22                          9698.22
ALBUS                    11908                           8514.30                          8512.78
AMORRIS                 310902                          53493.30                         53493.30
AMRTEST                     58                              7.08                             7.08
ANTEC64                   5512                            576.58                           576.58
ANTIETAM               2131824                         155222.58                        147673.32
...
```

- Note that each non-aggregate column in the `select` list has to appear in the `group by` clause

OCCUPANCY table example – ORDER BY

- Use the `ORDER BY` clause to sort the output results before they are displayed

  `ORDER BY col1 [ASC|DESC] [,col2 [ASC|DESC]] ...`

  – Specify column name or ordinal number, e.g., the first column (left to right) is **1**, the second column is **2**, etc.

  – If the column is unnamed, the ordinal number is required
    - Unnamed columns are columns whose value is calculated, e.g., the % USED column from prior examples or aggregate functions

  – Most columns are sorted in ascending order by default

OCCUPANCY table example – ORDER BY (*continued*)

- To wrap up this example, let's put the biggest users at the top of this list:

```
tsm: SS2_ADSM_GROUP_SERVER>select node_name, sum(num_files) as "Num. files", sum(physical_mb)
as "Physical MB", sum(logical_mb) as "Logical MB" from occupancy where type = 'Bkup' group by node_name
order by 4 desc

NODE_NAME           Num. files                      Physical MB                      Logical MB
------------------ ----------- --------------------------------- ---------------------------------
MONGO                  9631623                        3308407.05                       3284299.91
PLATO                  1036481                        1985495.08                       1984146.80
PERFDC                  537832                        1357331.88                       1354713.16
OTELLO                 1321371                         864444.26                        864057.15
STORMAN                5258006                         539721.85                        531174.71
TUCSVTLAB               748976                         325760.73                        324937.12
BROCKMAN               2028130                         271185.95                        271075.51
FRASERMC                703716                         196578.33                        195235.44
RAIBECK                 695075                         188982.31                        184589.51
TRIBLE                 1496072                         184948.98                        183984.71
HUSFELT                1466374                         183191.78                        181805.08
CKAETZ                  542598                         174908.72                        174832.95
...
```

- This sorts the output by "Logical MB" (the 4th column) in descending order

OCCUPANCY table example – how it works

- How it works:
  - Rows are returned from the OCCUPANCY table
  - `GROUP BY` puts the rows in sets according to the node name
  - `SUM` adds up the NUM_FILES, PHYSICAL_MB, and LOGICAL_MB values in each set
  - `ORDER BY` sorts the results by LOGICAL_MB
  - `SELECT` displays the resulting rows

Using a subquery in a predicate

- You can use a "subquery" as part of a predicate.
- For example, suppose you want to identify all nodes that are not associated with at least one schedule:

```
select node_name from nodes where node_name not in
   (select distinct node_name from associations)
```

- Note the use of the `DISTINCT` keyword.
  - Since a node can be associated with more than one schedule, `select node_name from associations` can return duplicate node names.
  - Using `DISTINCT` eliminates duplicate records from the result set.

The COLUMNS table

- Our brief introduction to `SELECT` is complete.
- Now let's look a closer look at the SYSCAT.COLUMNS table.

```
select * from columns where tabname='OCCUPANCY'
```

The COLUMNS table (*continued*)

```
tsm: SS2_ADSM_GROUP_SERVER>select * from columns where tabname='OCCUPANCY'

   TABSCHEMA: ADSM
     TABNAME: OCCUPANCY
     COLNAME: NODE_NAME
       COLNO: 1
INDEX_KEYSEQ: 1
 INDEX_ORDER: A
    TYPENAME: VARCHAR
      LENGTH: 64
       SCALE: 0
       NULLS: FALSE
     REMARKS: Node Name

   TABSCHEMA: ADSM
     TABNAME: OCCUPANCY
     COLNAME: TYPE
       COLNO: 2
INDEX_KEYSEQ:
 INDEX_ORDER:
    TYPENAME: VARCHAR
      LENGTH: 20
       SCALE: 0
       NULLS: FALSE
     REMARKS: Type
...
```

The COLUMNS table (*continued*)

- Note that each row in the COLUMNS table consists of the following columns:
  - TABSCHEMA: Either ADSM or SYSTEM.
    - SYSTEM (of which COLUMNS is a part) describes the structure of the tables.
    - The ADSM schema is comprised of the tables containing the real data of interest.
    - In general, this column is of little interest.
  - TABNAME: The name of the table to which the column belongs.
  - COLNAME: The name of the column.
  - COLNO: The ordinal number of the column within a table row. When your SELECT list is *, the columns are displayed from left to write according to the column number.

The COLUMNS table (*continued*)

- Note that each row in the COLUMNS table consists of the following columns:
  - INDEX_KEYSEQ: The ordinal number of the index key for which this column is used to index the table. If this column is not used to index the table, then INDEX_KEYSEQ is null.
    - For optimal performance, wherever possible put indexed keys at the beginning of `WHERE` criteria in lowest → highest order.
  - INDEX_ORDER: The collating order (A=Ascending or D=Descending) for the column if it is used to index the table. If this column is not used to index the table, then INDEX_ORDER is null.
  - TYPENAME: The data type for the column, e.g., character, numeric, timestamp, etc.

The COLUMNS table (*continued*)

- Note that each row in the COLUMNS table consists of the following columns:
  - LENGTH: The maximum width of the column
  - SCALE: For numeric values, the number of digits to the right of the decimal point
  - NULLS: Whether the column can be null
  - REMARKS: Brief description of the column's purpose

NULL columns

- In SQL, NULL is not equivalent to 0, empty string, FALSE, or other value.
- NULL means "unknown".
- WHERE predicates
- NULL columns, if ignored, can result in accurate results.

NULL columns (*continued*)

- Consider the following example, which identifies file spaces that have not been backed up in at least the last 30 days:

```
tsm: SS2_ADSM_GROUP_SERVER>select node_name, filespace_name,
date(backup_end) as "BACKUP END DATE" from filespaces
where backup_end < current_timestamp - 10 days

NODE_NAME            FILESPACE_NAME          BACKUP END DATE
-----------------    ------------------      ---------------
ADSMAPAR             \\apardude\c$                2005-03-14
ADSMPERF             /                            2005-06-30
ADSMPERF             /adsmperf_home               2005-06-30
AKOJENOV             \\aleko\c$                   2007-04-30
AKOJENOV             \\aleko\e$                   2007-04-29
...
```

NULL columns (*continued*)

- On the surface, it is a valid SELECT statement and seems to generate valid output.
- One problem that is non-obvious: if a file space for a node has never completed a backup, then the BACKUP_END column will be null.
- Since we are looking for file spaces that have not been backed up in the past 10 days, we would probably be interested in file spaces that have never been backed up.

NULL columns (*continued*)

- The solution is to also consider null BACKUP_END :

```
tsm: SS2_ADSM_GROUP_SERVER>select node_name, filespace_name,
date(backup_end) as "BACKUP END DATE" from filespaces
where backup_end is null or backup_end < current_timestamp - 10 days

NODE_NAME            FILESPACE_NAME          BACKUP END DATE
------------------   ------------------      ---------------
3A6929               \\ibm-130af90178f-
                      \c$
3A6929               \\n3a6929b\c$
ADSMAPAR             \\apardude\c$                2005-03-14
ADSMAPAR             \\branch\c$
ADSMAPAR             \\broyhill\c$
ADSMAPAR             \\jee\c$
ADSMPERF             /                            2005-06-30
ADSMPERF             /adsmperf_home               2005-06-30
ADSMPERF             /usr
AIX_CLIENT_BUILDS    /build
AIX_CLIENT_BUILDS    /build1
AKOJENOV             \\aleko\c$                   2007-04-30
AKOJENOV             \\aleko\e$                   2007-04-29
...
```

NULL columns (*continued*)

- Recommendations
  - If you think your results do not look quite right, check to see if null values might be a factor.
  - When in doubt, test for nulls.
  - Check the COLUMNS table for a given column's nullability.

```
tsm: SS2_ADSM_GROUP_SERVER>select nulls from columns
where tabname='FILESPACES' and colname='BACKUP_END'

NULLS
-----
 TRUE

tsm: SS2_ADSM_GROUP_SERVER>
```

Other considerations

- The SELECT command is a subset of SQL92.
- SELECT can be issued by any administrator.
- The following are not supported: UNION, INTERSECT, EXCEPT, and correlated subqueries.

Other considerations

- A minimum of one 4 MB database partition is required to use the SELECT command.
  - More free space may be required, depending on the amount of resource the query requires.
- Complex or long-running queries may affect server performance.
- The server prompts for confirmation if the query will require a significant amount of server time or other resources.

Other information sources

- The TSM Administrator's Reference
- Online help from the TSM command line administrator:
  - HELP SELECT
- SQL books
- Marist ADSM-L list server
  - Many users share their SQL commands

The End

Questions?